Distributed Cache Service

User Guide

Date **2024-07-26**

Contents

1 Service Overview

1.1 What Is DCS?

Distributed Cache Service (DCS) is an online, distributed, in-memory cache service compatible with Redis and Memcached. It is reliable, scalable, usable out of the box, and easy to manage, meeting your requirements for high read/write performance and fast data access.

- Usability out of the box

  DCS provides single-node, master/standby, and cluster instances with specifications ranging from 128 MB to 1024 GB. DCS instances can be created with just a few clicks on the console, without requiring you to prepare servers.

  DCS Redis 4.0/5.0/6.0 instances are containerized and can be created within seconds.

- Security and reliability

  Instance data storage and access are securely protected through security management services, including Identity and Access Management (IAM), Virtual Private Cloud (VPC), Cloud Eye, and Cloud Trace Service (CTS).

  Master/standby and cluster instances can be deployed within an availability zone (AZ) or across AZs.

- Auto scaling

  DCS instances can be scaled up or down online, helping you control costs based on service requirements.

- Easy management

  A web-based console is provided for you to perform various operations, such as restarting instances, modifying configuration parameters, and backing up and restoring data. RESTful application programming interfaces (APIs) are also provided for automatic instance management.

- Online migration

  You can create a data migration task on the console to import backup files or migrate data online.

DCS for Redis

Redis is a storage system that supports multiple types of data structures, including key-value pairs. It can be used in such scenarios as data caching, event publication/subscription, and high-speed queuing, as described in **Application Scenarios**. Redis is written in ANSI C, supporting direct read/write of **strings**, **hashes**, **lists**, **sets**, **sorted sets**, and **streams**. Redis works with an in-memory dataset which can be persisted on disk.

DCS Redis instances can be customized based on your requirements.

Table 1-1 DCS Redis instance configuration

Instance type	DCS for Redis provides the following types of instances to suit different service scenarios: Single-node: Suitable for caching temporary data in low reliability scenarios. Single-node instances support highly concurrent read/write operations, but do not support data persistence. Data will be deleted after instances are restarted. Master/standby: Each master/standby instance runs on two nodes (one master and one standby). The standby node replicates data synchronously from the master node. If the master node fails, the standby node automatically becomes the master node. Proxy Cluster: In addition to the native Redis cluster, a Proxy Cluster instance has proxies and load balancers. Load balancers implement load balancing. Different requests are distributed to different proxies to achieve high-concurrency. Each shard in the cluster has a master node and a standby node. If the master node is faulty, the standby node on the same shard is promoted to the master role to take over services. Redis Cluster: Each Redis Cluster instance consists of multiple **shards** and each shard includes a master node and multiple replicas (or no replica at all). Shards are not visible to you. If the master node fails, a replica on the same shard takes over services. You can split read and write operations by writing to the master node and reading from the replicas. This improves the overall cache read/write performance.
Instance specification	DCS for Redis provides instances of different specifications, ranging from 128 MB to 1024 GB.
Redis version	DCS instances are compatible with open-source Redis 3.0/4.0/5.0/6.0.
Underlying architecture	Deployed on large-specs VMs. 100,000 QPS at a single node.

High availability (HA) and DR	Master/standby and cluster DCS Redis instances can be deployed across AZs in the same region with physically isolated power supplies and networks.

For more information about open-source Redis, visit **https://redis.io/**.

DCS for Memcached

Memcached is an in-memory key-value caching system that supports read/write of simple strings. It is often used to cache backend database data to alleviate load on these databases and accelerate web applications. For details about its application scenarios, see **Memcached Application Scenarios**.

In addition to full compatibility with Memcached, DCS for Memcached provides the hot standby and data persistence.

Table 1-2 DCS Memcached instance configuration

Instance type	DCS for Memcached provides the following two types of instances to suit different service scenarios: Single-node: Suitable for caching temporary data in low reliability scenarios. Single-node instances support highly concurrent read/write operations, but do not support data persistence. Data will be deleted after instances are restarted. Master/standby: Each master/standby instance runs on two nodes (one master and one standby). The standby node replicates data synchronously from the master node, but does not support read/write operations. If the master node fails, the standby node automatically becomes the master node.
Memory	Specification of single-node or master/standby DCS Memcached instances: 2 GB, 4 GB, 8 GB, 16 GB, 32 GB, and 64 GB.
HA and DR	Master/Standby DCS Memcached instances can be deployed across AZs in the same region with physically isolated power supplies and networks.

For more information about open-source Memcached, visit **https://memcached.org/**.

1.2 Application Scenarios

Redis Application Scenarios

Many large-scale e-commerce websites and video streaming and gaming applications require fast access to large amounts of data that has simple data structures and does not need frequent join queries. In such scenarios, you can use

Redis to achieve fast yet inexpensive access to data. Redis enables you to retrieve data from in-memory data stores instead of relying entirely on slower disk-based databases. In addition, you no longer need to perform additional management tasks. These features make Redis an important supplement to traditional disk-based databases and a basic service essential for internet applications receiving high-concurrency access.

Typical application scenarios of DCS for Redis are as follows:

1. **E-commerce flash sales**

   E-commerce product catalogue, deals, and flash sales data can be cached to Redis.

   For example, the high-concurrency data access in flash sales can be hardly handled by traditional relational databases. It requires the hardware to have higher configuration such as disk I/O. By contrast, Redis supports 100,000 QPS per node and allows you to implement locking using simple commands such as **SET**, **GET**, **DEL**, and **RPUSH** to handle flash sales.

   For details about locking, see the "Implementing Distributed Locks" best practice.

2. **Live video commenting**

   In live streaming, online user, gift ranking, and bullet comment data can be stored as sorted sets in Redis.

   For example, bullet comments can be returned using the **ZREVRANGEBYSCORE** command. The **ZPOPMAX** and **ZPOPMIN** commands in Redis 5.0 can further facilitate message processing.

3. **Game leaderboard**

   In online gaming, the highest ranking players are displayed and updated in real time. The leaderboard ranking can be stored as sorted sets, which are easy to use with up to 20 commands.

   For details, see the "Ranking with Redis" best practice.

4. **Social networking comments**

   In web applications, queries of post comments often involve sorting by time in descending order. As comments pile up, sorting becomes less efficient.

   By using lists in Redis, a preset number of comments can be returned from the cache, rather than from disk, easing the load off the database and accelerating application responses.

Memcached Application Scenarios

Memcached is suitable for storing simple key-value data.

1. Web pages

   Caching static data such as HTML pages, Cascading Style Sheets (CSS), and images to DCS Memcached instances improves access performance of web pages.

2. Frontend database

   In dynamic systems such as social networking and blogging sites, write operations are far fewer than read operations such as querying users, friends, and articles. Such frequently access data can be cached in Memcached to reduce database load and improve performance.

The following data can be cached:

- Frequently accessed data that does not require real-time updates and can expire automatically

  Example: latest article lists and rankings. Although data is generated constantly, its impact on user experience is limited. Such data can be cached for a preset period of time and accessed from the database after this period. If web page editors want to view the latest ranking, a cache clearing or refreshing policy can be configured.

- Frequently accessed data that requires real-time updates

  Example: friend lists, article lists, and reading records. Such data can be cached to Memcached first, and then updated whenever changes (adding, modifying, and deleting data) occur.

3. Flash sales

   It is difficult for traditional databases to write an order placement operation during flash sales into the database, modify the inventory data, and ensure transaction consistency while ensuring uninterrupted user experience.

   Memcached **incr** and **decr** commands can be used to store inventory information and complete order placement in memory. Once an order is submitted, an order number is generated. Then, the order can be paid.

NOTE

Scenarios where Memcached is not suitable:

- The size of a single cache object is larger than 1 MB.

  Memcached cannot cache an object larger than 1 MB. In such cases, use Redis.

- The key contains more than 250 characters.

  To use Memcached in such a scenario, you can generate an MD5 hash for the key and cache the hash instead.

- High data reliability is required.

  Open-source Memcached does not provide data replication, backup, and migration, so data persistence is not supported.

  Master/Standby DCS Memcached instances support data persistence. For more information, contact technical support.

- Complex data structures and processing are required.

  Memcached supports only simple key-value pairs, and does not support complex data structures such as lists and sets, or complex operations such as sorting.

1.3 DCS Instance Types

1.3.1 Single-Node Redis

A single-node DCS Redis instance has only one node and does not support data persistence. They are suitable for cache services that do not require data reliability.

NOTE

- You cannot upgrade the Redis version of an instance. For example, a single-node DCS instance cannot be upgraded from Redis 4.0 to Redis 5.0. If you need Redis features of later versions, create a DCS Redis instance of a later version and then migrate data from the earlier instance to the new one.
- Single-node instances cannot ensure data persistence and do not support manual or scheduled data backup. Exercise caution before using them.

Features

1. Low system overhead and high QPS

   Single-node instances do not support data synchronization or data persistence, reducing system overhead and supporting higher concurrency. QPS of single-node DCS Redis instances reaches up to 100,000.
2. Process monitoring and automatic fault recovery

   With an HA monitoring mechanism, if a single-node DCS instance becomes faulty, a new process is started within 30 seconds to resume service provisioning.
3. Out-of-the-box usability and no data persistence

   Single-node DCS instances can be used out of the box because they do not involve data loading. If your service requires high QPS, you can warm up the data beforehand to avoid strong concurrency impact on the backend database.
4. Low-cost and suitable for development and testing

   Single-node instances are 40% cheaper than master/standby DCS instances, suitable for setting up development or testing environments.

In summary, single-node DCS instances support highly concurrent read/write operations, but do not support data persistence. Data will be deleted after instances are restarted. They are suitable for scenarios which do not require data persistence, such as database front-end caching, to accelerate access and ease the concurrency load off the backend. If the desired data does not exist in the cache, requests will go to the database. When restarting the service or the DCS instance, you can pre-generate cache data from the disk database to relieve pressure on the backend during startup.

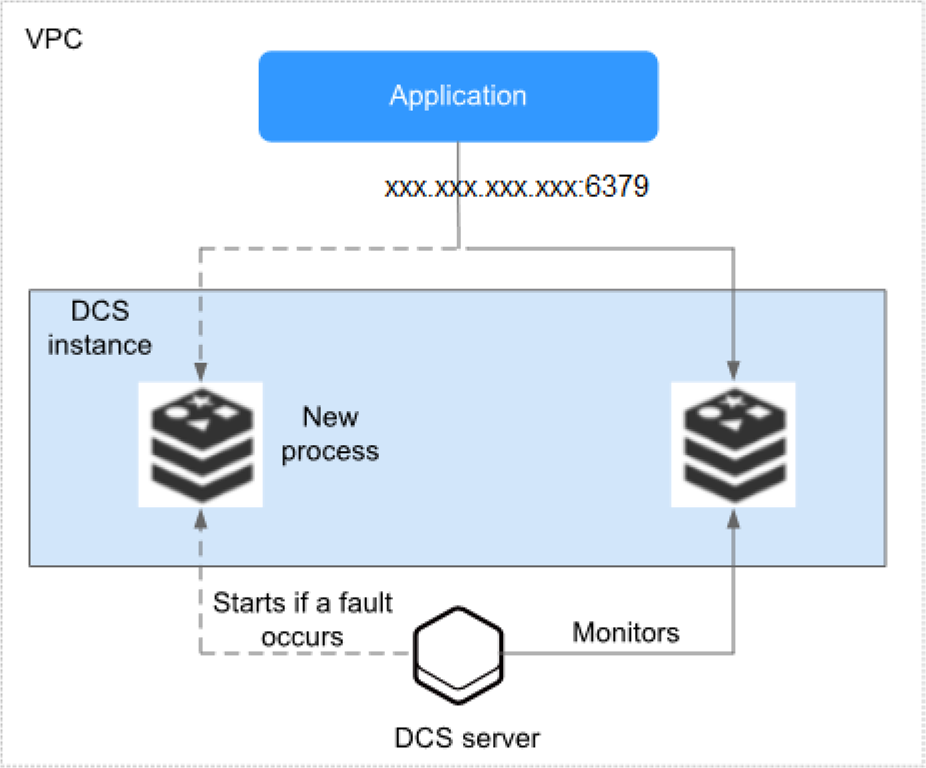
Architecture

Figure 1-1 shows the architecture of single-node DCS Redis instances.

NOTE

To access a DCS Redis 3.0 instance, you must use port 6379. To access a DCS Redis 4.0/5.0/6.0 instance, you can customize the port. If no port is specified, the default port 6379 will be used. In the following architecture, port 6379 is used. If you have customized a port, replace **6379** with the actual port.

Figure 1-1 Single-node DCS Redis instance architecture



Architecture description:

- **VPC**

  All server nodes of the instance run in the same VPC.

  NOTE

  For intra-VPC access, the client and the instance must be in the same VPC with specified security group rule configurations.

  For details, see **Security Group Configurations**.

- **Application**

  The client of the instance, which is the application running on an Elastic Cloud Server (ECS).

  DCS Redis instances are compatible with the Redis protocol, and can be accessed through open-source clients. For details about accessing DCS instances, see **Accessing an Instance**.

- **DCS instance**

  A single-node DCS instance, which has only one node and one Redis process.

  DCS monitors the availability of the instance in real time. If the Redis process becomes faulty, DCS starts a new process to resume service provisioning.

1.3.2 Master/Standby Redis

This section describes master/standby DCS Redis instances.

NOTE

You cannot upgrade the Redis version for an instance. For example, a master/standby DCS Redis 4.0 instance cannot be upgraded to a master/standby DCS Redis 5.0 instance. If your service requires the features of higher Redis versions, create a DCS Redis instance of a higher version and then migrate data from the old instance to the new one.

Features

Master/Standby DCS instances have higher availability and reliability than single-node DCS instances.

Master/Standby DCS instances have the following features:

1. **Data persistence and high reliability**

   By default, data persistence is enabled by both the master and the standby node of a master/standby instance.

   The standby node of a DCS Redis instance is invisible to you. Only the master node provides data read/write operations.

2. **Data synchronization**

   Data in the master and standby nodes is kept consistent through incremental synchronization.

   NOTE

   After recovering from a network exception or node fault, master/standby instances perform a full synchronization to ensure data consistency.

3. **Automatic master/standby switchover**

   If the master node becomes faulty, the instance is disconnected and unavailable for several seconds. The standby node takes over within 30 seconds without manual operations to resume stable services.

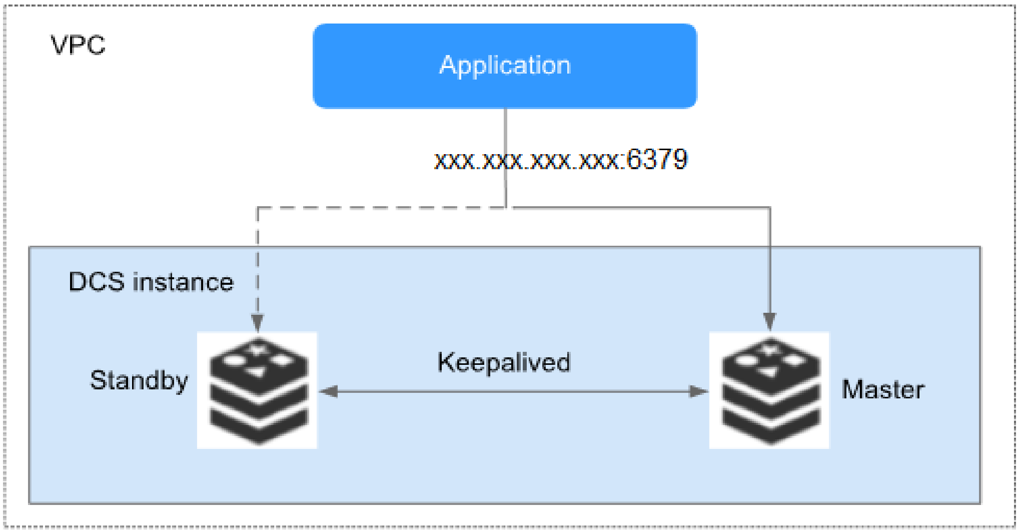
4. **DR policies**

   Each master/standby or cluster DCS instance can be deployed across AZs with physically isolated power supplies and networks. Applications can also be deployed across AZs to achieve high availability for both data and applications.

Architecture of DCS Redis 3.0 Instances

Figure 1-2 shows the architecture of master/standby DCS Redis instances.

Figure 1-2 Master/Standby DCS instance architecture



Architecture description:

- **VPC**

  All server nodes of the instance run in the same VPC.

  NOTE

  For intra-VPC access, the client and the instance must be in the same VPC with specified security group rule configurations.

  For details, see **Security Group Configurations**.

- **Application**

  The Redis client of the instance, which is the application running on the ECS.

  DCS Redis instances are compatible with the Redis protocol, and can be accessed through open-source clients. For details about accessing DCS instances, see **Accessing an Instance**.

- **DCS instance**

  Indicates a master/standby DCS instance which has a master node and a standby node. By default, data persistence is enabled and data is synchronized between the two nodes.

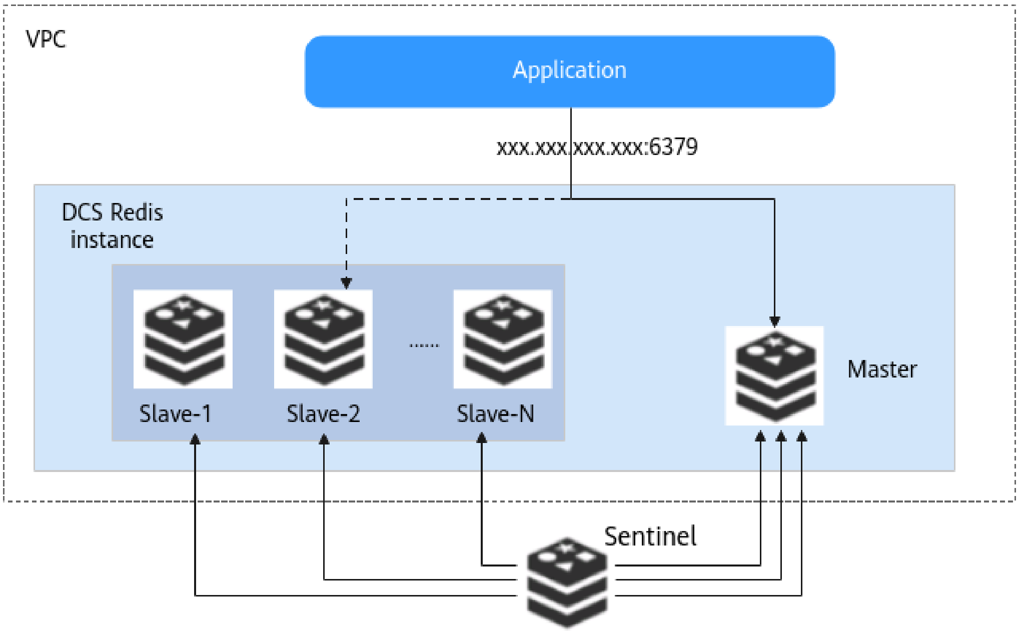
  DCS monitors the availability of the instance in real time. If the master node becomes faulty, the standby node becomes the master node and resumes service provisioning.

  DCS Redis 3.0 instances are accessed through port 6379 by default. Port customization is not supported.

Architecture of Master/Standby DCS Redis 4.0/5.0/6.0 Instances

The following figure shows the architecture of a master/standby DCS Redis 4.0/5.0/6.0 instance.

Figure 1-3 Architecture of a master/standby DCS Redis 4.0/5.0/6.0 instance



Architecture description:

1. Master/standby DCS Redis 4.0/5.0/6.0 instances support Sentinels. Sentinels monitor the running status of the master and standby nodes. If the master node becomes faulty, a failover will be performed.

   Sentinels are invisible to you and is used only in the service. For details about Sentinel, see **What Is Sentinel?**
2. A standby node has the same specifications as a master node. A master/standby instance consists of a pair of master and standby nodes by default.
3. To access a DCS Redis 4.0/5.0/6.0 instance, you can customize the port. If no port is specified, the default port 6379 will be used. In the architecture diagram, port 6379 is used. If you have customized a port, replace **6379** with the actual port.

1.3.3 Proxy Cluster Redis

DCS for Redis provides Proxy Cluster instances, which use Linux Virtual Server (LVS) and proxies to achieve high availability. Proxy Cluster instances have the following features:

- The client is decoupled from the cloud service.
- They support millions of concurrent requests, equivalent to Redis Cluster instances.
- A wide range of memory specifications adapt to different scenarios.

NOTE

- A Proxy Cluster instance can be connected in the same way that a single-node or master/standby instance is connected, without any special settings on the client. You can use the IP address of the instance, and do not need to know or use the proxy or shard addresses.

Proxy Cluster DCS Redis 3.0 Instances

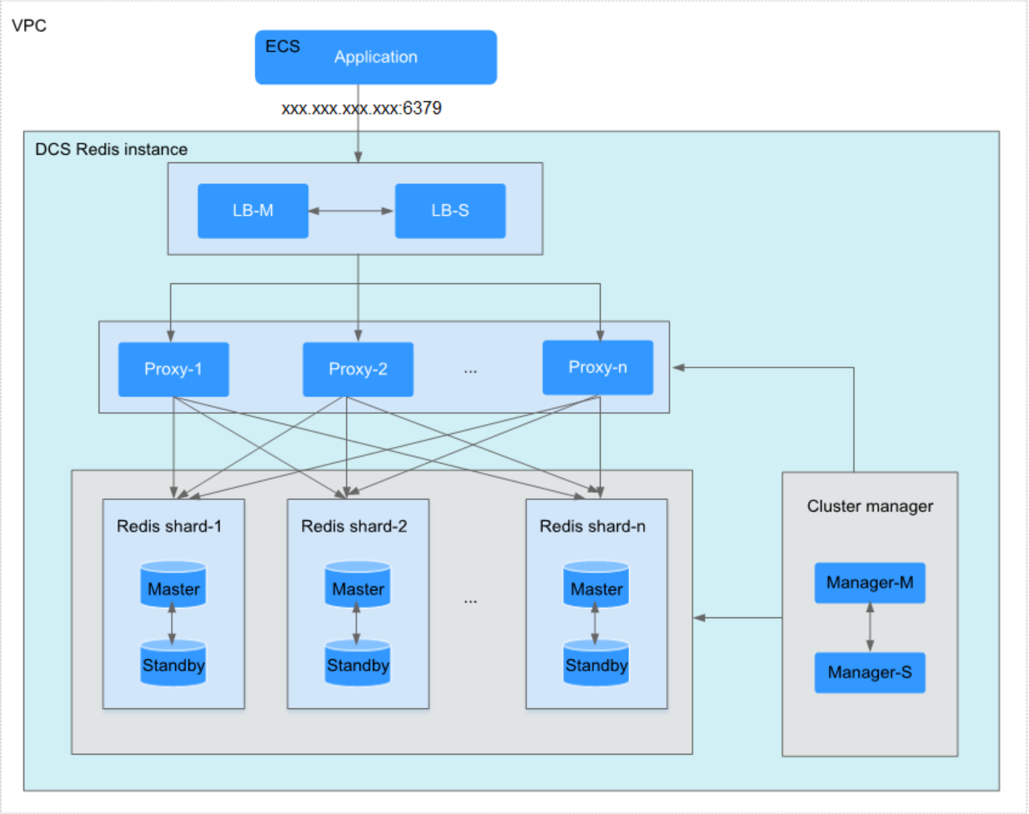
Proxy Cluster DCS Redis 3.0 instances are compatible with **codis**. The specifications range from 64 GB to 1024 GB, meeting requirements for **millions of concurrent connections** and **massive data cache**. Distributed data storage and access is implemented by DCS, without requiring development or maintenance.

Each Proxy Cluster instance consists of load balancers, proxies, cluster managers, and **shards**.

Table 1-3 Total memory, proxies, and shards of Proxy Cluster DCS Redis 3.0 instances

Total Memory	Proxies	Shards
64 GB	3	8
128 GB	6	16
256 GB	8	32

Figure 1-4 Proxy Cluster DCS Redis instance architecture



Architecture description:

- **VPC**
  All server nodes of the cluster instance run in the same VPC.

NOTE

For intra-VPC access, the client and the instance must be in the same VPC with specified security group rule configurations.

For details, see **Security Group Configurations**.

- **Application**

  The client used to access the instance.

  DCS Redis instances can be accessed through open-source clients. For details about accessing DCS instances, see **Accessing an Instance**.

- **LB-M/LB-S**

  The load balancers, which are deployed in master/standby HA mode. The connection addresses (**IP address:Port**) of the cluster DCS Redis instance are the addresses of the load balancers.

- **Proxy**

  The proxy server used to achieve high availability and process high-concurrency client requests.

  You can connect to a Proxy Cluster instance at the IP addresses of its proxies.

- **Redis shard**

  A shard of the cluster.

  Each shard consists of a pair of master/standby nodes. If the master node becomes faulty, the standby node automatically takes over cluster services.

  If both the master and standby nodes of a shard are faulty, the cluster can still provide services but the data on the faulty shard is inaccessible.

- **Cluster manager**

  The cluster configuration managers, which store configurations and partitioning policies of the cluster. You cannot modify the information about the configuration managers.

1.3.4 Redis Cluster

Redis Cluster DCS instances use the native distributed implementation of Redis. Redis Cluster instances have the following features:

- They are compatible with native Redis clusters.
- They inherit the smart client design from Redis.
- They deliver many times higher performance than master/standby instances.

Read/write splitting is supported by configuring the client for Redis Cluster instances but is not supported for Proxy Cluster instances. **Read more** about DCS's support for read/write splitting.

Redis Cluster

The Redis Cluster instance type provided by DCS is compatible with the **native Redis Cluster**, which uses smart clients and a distributed architecture to perform sharding.

Table 1-4 lists the shard specification for different instance specifications.

Size of a shard = Instance specification/Number of shards. For example, if a 48 GB instance has 6 shards, the size of each shard is 48 GB/6 = 8 GB.

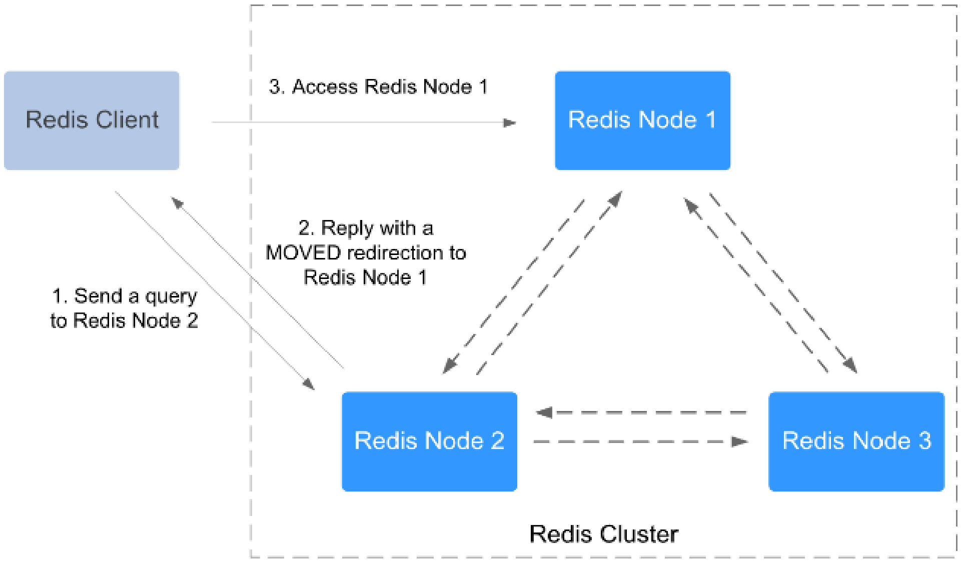
Table 1-4 Specifications of Redis Cluster DCS instances

Total Memory	Shards
4 GB/8 GB/16 GB/24 GB/32 GB	3
48 GB	6
64 GB	8
96 GB	12
128 GB	16
192 GB	24
256 GB	32
384 GB	48
512 GB	64
768 GB	96
1024 GB	128

- Distributed architecture

  Any node in a Redis Cluster can receive requests. Received requests are then redirected to the right node for processing. Each node consists of a subset of one master and one (by default) or multiple replicas. The master or replica roles are determined through an election algorithm.

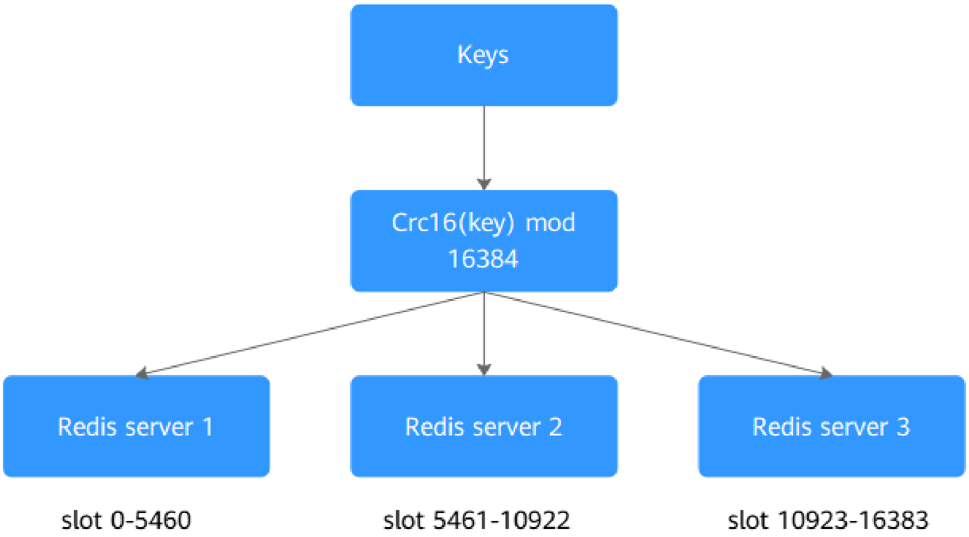
  Figure 1-5 Distributed architecture of Redis Cluster



- Presharding

  There are 16,384 hash slots in each Redis Cluster. The mapping between hash slots and Redis nodes is stored in Redis Servers. To compute what is the hash slot of a given key, simply take the CRC16 of the key modulo 16384. Example command output

Figure 1-6 Redis Cluster presharding



1.3.5 Comparing DCS Redis Instance Types

Table 1-5 describes the differences between different Redis instance types in terms of features and commands.

Table 1-5 Differences between DCS instance types

Item	Single-Node or Master/ Standby	Proxy Cluster	Redis Cluster
Redis version compatibility	Redis 3.0/4.0/5.0/6.0. You can select a version when creating an instance.	Redis 3.0	Redis 4.0/5.0. You can select a version when creating an instance.
Support	● Keyspace notifications ● Pipelining	● Pipelining, **MSET** command, and **MGET** command ● **SCAN** command, **KEYS** command, and Redis Slow Log ● Pub/Sub	● Keyspace notifications ● **BRPOP**, **BLPOP**, and **BRPOPLPUSH** commands ● Pub/Sub

Item	Single-Node or Master/ Standby	Proxy Cluster	Redis Cluster
Restrictions	Single-node instances do not support data persistence, backup, or restoration.	● LUA script is restricted: All keys must be in the same hash slot to avoid errors. **Hash tags** are recommended. ● Some commands that contain multiple keys require that the keys must be in the same hash slot to avoid errors. **Hash tags** are recommended. For details about these multi-key commands, see **Table 1-38**. ● Keyspace notifications are not supported.	● LUA script is restricted: All keys must be in the same hash slot. **Hash tags** are recommended. ● The client SDK must support Redis Cluster and be able to process MOVED errors. ● When you are using pipelining, **MSET** command, or **MGET** command, all keys must be in the same hash slot to avoid errors. **Hash tags** are recommended. ● When using keyspace notifications, establish connections with every Redis server and process events on each connection. ● When using a traversing or global command such as **SCAN** and **KEYS**, run the command on each Redis server.
Client	Any Redis client	Any Redis client (no need to support the Redis Cluster protocol)	Any client that supports the Redis Cluster protocol
Disabled commands	**Command Compatibility** lists disabled commands.	**Command Compatibility** lists disabled commands.	**Command Compatibility** lists disabled commands.

Item	Single-Node or Master/ Standby	Proxy Cluster	Redis Cluster
Replicas	A single-node instance has only one replica. By default, a master/standby instance has two replicas, with one of them being the master. When creating a master/ standby DCS Redis 4.0 or 5.0 instance, you can customize the number of replicas, with one of them being the master. Currently, the number of replicas cannot be customized for master/ standby DCS Redis 3.0 and 6.0 instances.	Each shard in the cluster has and can only have two replicas, with one of them being the master.	By default, each shard in a cluster has two replicas. The number of replicas on each shard can be customized, with one of them being the master. When creating an instance, you can set the replica quantity to one, indicating that the instance only has the master node. In this case, high data reliability cannot be ensured.

1.3.6 Single-Node Memcached

This section describes the features and architecture of single-node DCS Memcached instances.

Features

1. Low system overhead and high QPS

   Single-node instances do not support data synchronization or data persistence, reducing system overhead and supporting higher concurrency. QPS of single-node DCS Memcached instances reaches up to 100,000.

2. Process monitoring and automatic fault recovery

   With an HA monitoring mechanism, if a single-node DCS instance becomes faulty, a new process is started within 30 seconds to resume service provisioning.

3. Out-of-the-box usability and no data persistence

   Single-node DCS instances can be used out of the box because they do not involve data loading. If your service requires high QPS, you can warm up the data beforehand to avoid strong concurrency impact on the backend database.

4. Low-cost and suitable for development and testing

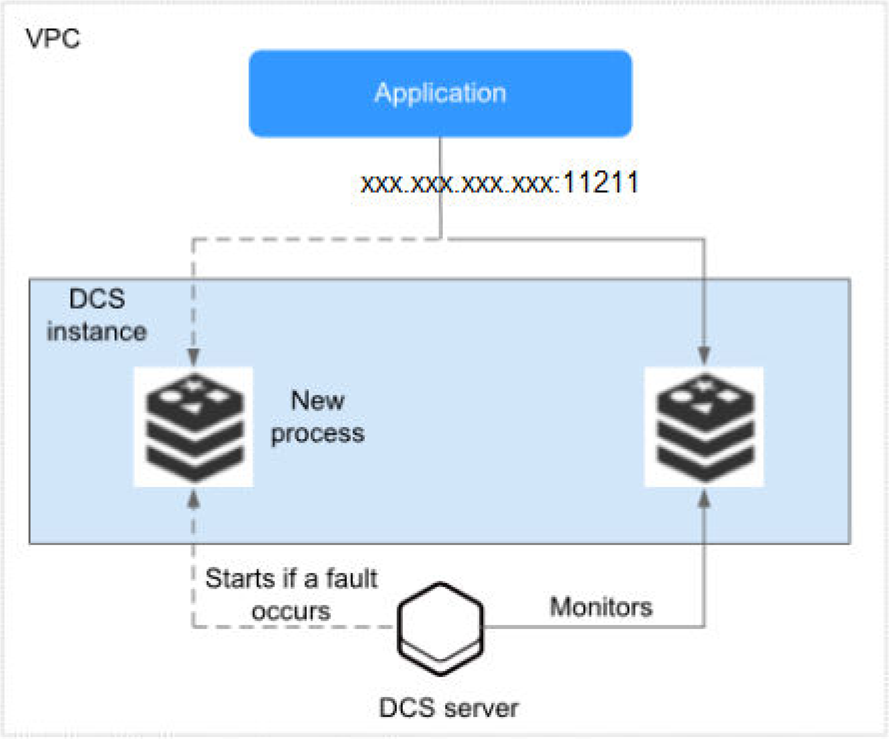
   Single-node instances are 40% cheaper than master/standby DCS instances, suitable for setting up development or testing environments.

In summary, single-node DCS instances support highly concurrent read/write operations, but do not support data persistence. Data will be deleted after instances are restarted. They are suitable for scenarios which do not require data persistence, such as database front-end caching, to accelerate access and ease the concurrency load off the backend. If the desired data does not exist in the cache, requests will go to the database. When restarting the service or the DCS instance, you can pre-generate cache data from the disk database to relieve pressure on the backend during startup.

Architecture

Figure 1-7 shows the architecture of single-node DCS Memcached instances.

Figure 1-7 Single-node DCS instance architecture



Architecture description:

- **VPC**

  The VPC where all nodes of the instance are run.

NOTE

For intra-VPC access, the client and the instance must be in the same VPC with specified security group rule configurations.

For details, see **Security Group Configurations**.

- **Application**

  The client of the instance, which is the application running on an Elastic Cloud Server (ECS).

  DCS Memcached instances are compatible with the Memcached protocol, and can be accessed through open-source clients. For examples of accessing DCS instances, see **Accessing a DCS Memcached Instance (Discontinued)**.

- **DCS instance**

  A single-node DCS instance, which has only one node and one Memcached process.

  DCS monitors the availability of the instance in real time. If the Memcached process becomes faulty, DCS starts a new process to resume service provisioning.

  Use port 11211 to access a DCS Memcached instance.

1.3.7 Master/Standby Memcached

This section describes master/standby DCS Memcached instances.

Features

Master/Standby instances have higher availability and reliability than single-node instances.

Master/Standby DCS Memcached instances have the following features:

1. **Data persistence and high reliability**

   By default, data persistence is enabled by both the master and the standby node of a master/standby DCS Memcached instance. In addition, data persistence is supported to ensure high data reliability.

   The standby node of a DCS Memcached instance is invisible to you. Only the master node provides data read/write operations.

2. **Data synchronization**

   Data in the master and standby nodes is kept consistent through incremental synchronization.

   NOTE

   After recovering from a network exception or node fault, master/standby instances perform a full synchronization to ensure data consistency.

3. **Automatic master/standby switchover**

   If the master node becomes faulty, the standby node takes over within 30 seconds, without requiring any service interruptions or manual operations.

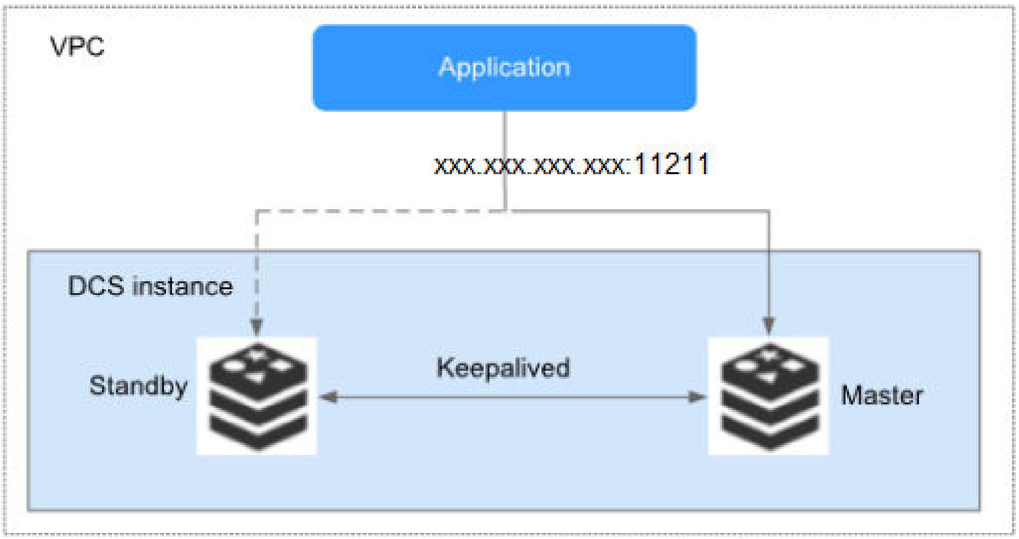
4. **Multiple DR policies**

   Each master/standby instance can be deployed across AZs with physically isolated power supplies and networks. Applications can also be deployed across AZs to achieve HA for both data and applications.

Architecture of Master/Standby DCS Memcached Instances

Figure 1-8 shows the architecture of master/standby DCS Memcached instances.

Figure 1-8 Master/Standby DCS Memcached instance architecture



Architecture description:

- **VPC**

  The VPC where all nodes of the instance are run.

  NOTE

  For intra-VPC access, the client and the instance must be in the same VPC with specified security group rule configurations.

  For details, see **Security Group Configurations**.

- **Application**

  The Memcached client of the instance, which is the application running on the ECS.

  DCS Memcached instances are compatible with the Memcached protocol, and can be accessed through open-source clients. For examples of accessing DCS instances, see **Accessing a DCS Memcached Instance (Discontinued)**.

- **DCS instance**

  Indicates a master/standby DCS instance which has a master node and a standby node. By default, data persistence is enabled and data is synchronized between the two nodes.

  DCS monitors the availability of the instance in real time. If the master node becomes faulty, the standby node becomes the master node and resumes service provisioning.

  Use port 11211 to access a DCS Memcached instance.

1.4 DCS Instance Specifications

1.4.1 Redis 3.0 Instance Specifications

This section describes DCS Redis 3.0 instance specifications, including the total memory, available memory, maximum number of connections allowed, maximum/assured bandwidth, and reference performance.

The following metrics are related to the instance specifications:

- Used memory: You can check the memory usage of an instance by viewing the **Memory Usage** and **Used Memory** metrics.
- Maximum connections: The maximum number of connections allowed is the maximum number of clients that can be connected to an instance. To check the number of connections to an instance, view the **Connected Clients** metric.
- QPS represents queries per second, which is the number of commands processed per second.

NOTE

- Single-node, master/standby, and Proxy Cluster types are available.
- Only the x86 architecture is supported. The Arm architecture is not supported.

Single-Node Instances

For each single-node DCS Redis instance, the available memory is less than the total memory because some memory is reserved for system overheads, as shown in the following table.

Table 1-6 Specifications of single-node DCS Redis 3.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
2	1.5	5000/50,000	42/512	50,000	dcs.single_node
4	3.2	5000/50,000	64/1536	100,000	dcs.single_node
8	6.8	5000/50,000	64/1536	100,000	dcs.single_node
16	13.6	5000/50,000	85/3072	100,000	dcs.single_node
32	27.2	5000/50,000	85/3072	100,000	dcs.single_node
64	58.2	5000/60,000	128/5120	100,000	dcs.single_node

Master/Standby Instances

For each master/standby DCS Redis instance, the available memory is less than that of a single-node DCS Redis instance because some memory is reserved for data persistence, as shown in the following table. The available memory of a master/standby instance can be adjusted to support background tasks such as data persistence and master/standby synchronization.

Table 1-7 Specifications of master/standby DCS Redis 3.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/ Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
2	1.5	5000/50,000	42/512	50,000	dcs.master_standby
4	3.2	5000/50,000	64/1536	100,000	dcs.master_standby
8	6.4	5000/50,000	64/1536	100,000	dcs.master_standby
16	12.8	5000/50,000	85/3072	100,000	dcs.master_standby
32	25.6	5000/50,000	85/3072	100,000	dcs.master_standby
64	51.2	5000/60,000	128/5120	100,000	dcs.master_standby

Proxy Cluster Instances

In addition to larger memory, cluster instances feature more connections allowed, higher bandwidth allowed, and more QPS than single-node and master/standby instances.

Table 1-8 Specifications of Proxy Cluster DCS Redis 3.0 instances

Specification (GB)	Available Memory (GB)	Max. Connections (Default/ Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
64	64	90,000/90,000	600/5120	500,000	dcs.cluster

Specification (GB)	Available Memory (GB)	Max. Connections (Default/ Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
128	128	180,000/180,000	600/5120	500,000	dcs.cluster
256	256	240,000/240,000	600/5120	500,000	dcs.cluster

1.4.2 Redis 4.0 and 5.0 Instance Specifications

This section describes DCS Redis 4.0 and 5.0 instance specifications, including the total memory, available memory, maximum number of connections allowed, maximum/assured bandwidth, and reference performance.

The following metrics are related to the instance specifications:

- Used memory: You can check the memory usage of an instance by viewing the **Memory Usage** and **Used Memory** metrics.
- Maximum connections: The maximum number of connections allowed is the maximum number of clients that can be connected to an instance. To check the number of connections to an instance, view the **Connected Clients** metric. After an instance is created, you can change the maximum number of connections of the instance by modifying the **maxclients** parameter on the **Instance Configuration** > **Parameters** page on the console. This parameter cannot be modified for Proxy Cluster instances.
- QPS represents queries per second, which is the number of commands processed per second.
- Bandwidth: You can view the **Flow Control Times** metric to check whether the bandwidth has exceeded the limit.

NOTE

- Single-node, master/standby, and Redis Cluster types are available.
- Only the x86 architecture is supported. The Arm architecture is not supported.

Single-Node Instances

Table 1-9 Specifications of single-node DCS Redis 4.0 or 5.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/ Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
0.125	0.125	10,000/10,000	40/40	80,000	redis.single.xu1.tiny.128
0.25	0.25	10,000/10,000	80/80	80,000	redis.single.xu1.tiny.256
0.5	0.5	10,000/10,000	80/80	80,000	redis.single.xu1.tiny.512
1	1	10,000/50,000	80/80	80,000	redis.single.xu1.large.1
2	2	10,000/50,000	128/128	80,000	redis.single.xu1.large.2
4	4	10,000/50,000	192/192	80,000	redis.single.xu1.large.4
8	8	10,000/50,000	192/192	100,000	redis.single.xu1.large.8
16	16	10,000/50,000	256/256	100,000	redis.single.xu1.large.16
24	24	10,000/50,000	256/256	100,000	redis.single.xu1.large.24
32	32	10,000/50,000	256/256	100,000	redis.single.xu1.large.32
48	48	10,000/50,000	256/256	100,000	redis.single.xu1.large.48

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/ Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
64	64	10,000/50,000	384/384	100,000	redis.single.xu1.large.64

Master/Standby Instances

By default, a master/standby instance has two replicas (including the master). There is one master node.

Number of IP addresses occupied by a master/standby instance = Number of master nodes x Number of replicas. For example:

2 replicas: Number of occupied IP addresses = 1 x 2 = 2

3 replicas: Number of occupied IP addresses = 1 x 3 = 3

The following table lists the specification codes (**spec_code**) when there are two default replicas. Change the replica quantity in the specification codes based on the actual number of replicas. For example, if an 8 GB master/standby x86-based instance has two replicas, its specification code is redis.ha.xu1.large. **r2**.8. If it has three replicas, its specification code is redis.ha.xu1.large. **r3**.8.

Table 1-10 Specifications of master/standby DCS Redis 4.0 or 5.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/ Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
0.125	0.125	10,000/10,000	40/40	80,000	redis.ha.xu1.tiny.r2.128
0.25	0.25	10,000/10,000	80/80	80,000	redis.ha.xu1.tiny.r2.256
0.5	0.5	10,000/10,000	80/80	80,000	redis.ha.xu1.tiny.r2.512
1	1	10,000/50,000	80/80	80,000	redis.ha.xu1.large.r2.1
2	2	10,000/50,000	128/128	80,000	redis.ha.xu1.large.r2.2

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/ Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
4	4	10,000/50,000	192/192	80,000	redis.ha.xu1.large.r2.4
8	8	10,000/50,000	192/192	100,000	redis.ha.xu1.large.r2.8
16	16	10,000/50,000	256/256	100,000	redis.ha.xu1.large.r2.16
24	24	10,000/50,000	256/256	100,000	redis.ha.xu1.large.r2.24
32	32	10,000/50,000	256/256	100,000	redis.ha.xu1.large.r2.32
48	48	10,000/50,000	256/256	100,000	redis.ha.xu1.large.r2.48
64	64	10,000/50,000	384/384	100,000	redis.ha.xu1.large.r2.64

Redis Cluster Instances

In addition to larger memory, Redis Cluster instances feature more connections allowed, higher bandwidth allowed, and more QPS than single-node and master/standby instances.

- Specification name: The following table only lists the specification names of 2-replica x86-based instances. The specification names reflect the number of replicas, for example, redis.cluster.xu1.large.**r2**.8 (x86 | 2 replicas | 8 GB) and redis.cluster.xu1.large.**r3**.8 (x86 | 3 replicas | 8 GB).
- IP addresses: Number of occupied IP addresses = Number of shards x Number of replicas. For example:

  4 GB | Redis Cluster | 3 replicas: Number of occupied IP addresses = 3 x 3 = 9
- Available memory per node = Instance available memory/Master node quantity. For example:

  A 24 GB instance has 24 GB available memory and 3 master nodes. The available memory per node is 24/3 = 8 GB.
- Maximum connections per node = Instance maximum connections/Master node quantity. For example:

  A 4 GB instance has 3 master nodes and the maximum connections limit is 150,000. The maximum connections limit per node = 150,000/3 = 50,000.

Table 1-11 Specifications of Redis Cluster DCS Redis 4.0 or 5.0 instances

Total Memory (GB)	Available Memory (GB)	Shards (Master Nodes)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
4	4	3	30,000 /150,000	2304/2304	240,000	redis.cluster.xu1.large.r2.4
8	8	3	30,000 /150,000	2304/2304	240,000	redis.cluster.xu1.large.r2.8
16	16	3	30,000 /150,000	2304/2304	240,000	redis.cluster.xu1.large.r2.16
24	24	3	30,000 /150,000	2304/2304	300,000	redis.cluster.xu1.large.r2.24
32	32	3	30,000 /150,000	2304/2304	300,000	redis.cluster.xu1.large.r2.32
48	48	6	60,000 /300,000	4608/4608	> 300,000	redis.cluster.xu1.large.r2.48
64	64	8	80,000 /400,000	6144/6144	500,000	redis.cluster.xu1.large.r2.64
96	96	12	120,000 /600,000	9216/9216	> 500,000	redis.cluster.xu1.large.r2.96
128	128	16	160,000 /800,000	12,288/12,288	1,000,000	redis.cluster.xu1.large.r2.128

Total Memory (GB)	Available Memory (GB)	Shards (Master Nodes)	Max. Connections (Default/Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
192	192	24	240,000 / 1,200,000	18,432/18,432	> 1,000,000	redis.cluster.xu1.large.r2.192
256	256	32	320,000 / 1,600,000	24,576/24,576	> 2,000,000	redis.cluster.xu1.large.r2.256
384	384	48	480,000 / 2,400,000	36,864/36,864	> 2,000,000	redis.cluster.xu1.large.r2.384
512	512	64	640,000 / 3,200,000	49,152/49,152	> 2,000,000	redis.cluster.xu1.large.r2.512
768	768	96	960,000 / 4,800,000	73,728/73,728	> 2,000,000	redis.cluster.xu1.large.r2.768
1024	1024	128	1,280,000 / 6,400,000	98,304/98,304	> 2,000,000	redis.cluster.xu1.large.r2.1024

1.4.3 Redis 6.0 Instance Specifications

This section describes DCS Redis 6.0 instance specifications, including the total memory, available memory, maximum number of connections, maximum/assured bandwidth, and reference performance.

The following metrics are related to the instance specifications:

- Used memory: You can check the memory usage of an instance by viewing the **Memory Usage** and **Used Memory** metrics.
- Maximum connections: The maximum number of connections allowed is the maximum number of clients that can be connected to an instance. To check

the number of connections to an instance, view the **Connected Clients** metric.

- QPS represents queries per second, which is the number of commands processed per second.
- Bandwidth: You can view the **Flow Control Times** metric to check whether the bandwidth has exceeded the limit.

Currently, DCS for Redis 6.0 supports single-node and master/standby instances based on x86 CPUs.

Single-Node

Table 1-12 Specifications of single-node DCS Redis 6.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
0.125	0.125	10,000/10,000	40/40	80,000	redis.single.xu1.tiny.128
0.25	0.25	10,000/10,000	80/80	80,000	redis.single.xu1.tiny.256
0.5	0.5	10,000/10,000	80/80	80,000	redis.single.xu1.tiny.512
1	1	10,000/50,000	80/80	80,000	redis.single.xu1.large.1
2	2	10,000/50,000	128/128	80,000	redis.single.xu1.large.2
4	4	10,000/50,000	192/192	80,000	redis.single.xu1.large.4
8	8	10,000/50,000	192/192	100,000	redis.single.xu1.large.8
16	16	10,000/50,000	256/256	100,000	redis.single.xu1.large.16
24	24	10,000/50,000	256/256	100,000	redis.single.xu1.large.24
32	32	10,000/50,000	256/256	100,000	redis.single.xu1.large.32
48	48	10,000/50,000	256/256	100,000	redis.single.xu1.large.48
64	64	10,000/50,000	384/384	100,000	redis.single.xu1.large.64

Master/Standby

Table 1-13 Specifications of master/standby DCS Redis 6.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
0.125	0.125	10,000/10,000	40/40	80,000	redis.ha.xu1.tiny.r2.128
0.25	0.25	10,000/10,000	80/80	80,000	redis.ha.xu1.tiny.r2.256
0.5	0.5	10,000/10,000	80/80	80,000	redis.ha.xu1.tiny.r2.512
1	1	10,000/50,000	80/80	80,000	redis.ha.xu1.large.r2.1
2	2	10,000/50,000	128/128	80,000	redis.ha.xu1.large.r2.2
4	4	10,000/50,000	192/192	80,000	redis.ha.xu1.large.r2.4
8	8	10,000/50,000	192/192	100,000	redis.ha.xu1.large.r2.8
16	16	10,000/50,000	256/256	100,000	redis.ha.xu1.large.r2.16
24	24	10,000/50,000	256/256	100,000	redis.ha.xu1.large.r2.24
32	32	10,000/50,000	256/256	100,000	redis.ha.xu1.large.r2.32
48	48	10,000/50,000	256/256	100,000	redis.ha.xu1.large.r2.48
64	64	10,000/50,000	384/384	100,000	redis.ha.xu1.large.r2.64

1.4.4 Memcached Instance Specifications

This section describes DCS Memcached instance specifications, including the total memory, available memory, maximum number of connections allowed, maximum/assured bandwidth, and reference performance.

Maximum connections: The maximum number of connections allowed is the maximum number of clients that can be connected to an instance. To check the number of connections to an instance, view the **Connected Clients** metric.

QPS represents queries per second, which is the number of commands processed per second.

NOTE

DCS Memcached instances are available in single-node and master/standby types.

Single-Node Instances

For each single-node DCS Memcached instance, the available memory is less than the total memory because some memory is reserved for system overheads, as shown in **Table 1-14**.

Table 1-14 Specifications of single-node DCS Memcached instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)
2	1.5	5000/50,000	42/128	50,000
4	3.2	5000/50,000	64/192	100,000
8	6.8	5000/50,000	64/192	100,000
16	13.6	5000/50,000	85/256	100,000
32	27.2	5000/50,000	85/256	100,000
64	58.2	5000/50,000	128/384	100,000

Master/Standby Instances

For each master/standby DCS Memcached instance, the available memory is less than the total memory because some memory is reserved for data persistence, as shown in **Table 1-15**. The available memory of a master/standby instance can be adjusted to support background tasks such as data persistence and master/standby synchronization.

Table 1-15 Specifications of master/standby DCS Memcached instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)
2	1.5	5000/50,000	42/128	50,000
4	3.2	5000/50,000	64/192	100,000
8	6.8	5000/50,000	64/192	100,000

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)
16	13.6	5000/50,000	85/256	100,000
32	27.2	5000/50,000	85/256	100,000
64	58.2	5000/50,000	128/384	100,000

1.5 Command Compatibility

1.5.1 Redis 3.0 Commands

DCS for Redis 3.0 is developed based on Redis 3.0.7 and is compatible with open-source protocols and commands.

This section describes DCS for Redis 3.0's compatibility with Redis commands, including supported commands, disabled commands, unsupported scripts and commands of later Redis versions, and restrictions on command usage. For more information about the command syntax, visit the **Redis official website**.

DCS for Redis instances support most Redis commands, which are listed in **Commands Supported by DCS for Redis 3.0**. Any client compatible with the Redis protocol can access DCS.

- For security purposes, some Redis commands are disabled in DCS, as listed in **Commands Disabled by DCS for Redis 3.0**.
- Some Redis commands are supported by cluster DCS instances for multi-key operations in the same slot. For details, see **Command Restrictions**.
- Some Redis commands have usage restrictions, which are described in **Other Command Usage Restrictions**.

Commands Supported by DCS for Redis 3.0

The following lists commands supported by DCS for Redis 3.0.

NOTE

- Commands available since later Redis versions are not supported by earlier-version instances. Run a command on redis-cli to check whether it is supported by DCS for Redis. If the message "(error) ERR unknown command" is returned, the command is not supported.
- The following commands listed in the tables are not supported by Proxy Cluster instances:
  - **List** group: **BLPOP**, **BRPOP**, and **BRPOPLRUSH**
  - **CLIENT** commands in the **Server** group: **CLIENT KILL**, **CLIENT GETNAME**, **CLIENT LIST**, **CLIENT SETNAME**, **CLIENT PAUSE**, and **CLIENT REPLY**.
  - **Server** group: **MONITOR**
  - **Key** group: **RANDOMKE** (for old Proxy Cluster instances)

Table 1-16 Commands supported by DCS Redis 3.0 instances (1)

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	KEYS
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT KILL
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYCORE	CLIENT GETNAME
RENAMENX	INCR	HSET	LREM	SPOP	ZREVRANGE	CLIENT SETNAME
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	CONFIG GET
SORT	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	MONITOR

Keys	String	Hash	List	Set	Sorted Set	Server
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	SLOWLOG
TYPE	MSET	-	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	ROLE
SCAN	MSETNX	-	RPOPLPUSH	SSCAN	ZINTERSTORE	-
OBJECT	PSETEX	-	RPUSH	-	ZSCAN	-
-	SET	-	RPUSHX	-	ZRANGEBYLEX	-
-	SETBIT	-	-	-	-	-
-	SETEX	-	-	-	-	-
-	SETNX	-	-	-	-	-
-	SETRANGE	-	-	-	-	-
-	STRLEN	-	-	-	-	-

Table 1-17 Commands supported by DCS Redis 3.0 instances (2)

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST
-	SUBSCRIBE	WATCH	SELECT	SCRIPT KILL	GEORADIUS
-	UNSUBSCRIBE	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER

Commands Disabled by DCS for Redis 3.0

The following lists commands disabled by DCS for Redis 3.0.

Table 1-18 Redis commands disabled in single-node and master/standby DCS Redis 3.0 instances

Keys	Server
MIGRATE	SLAVEOF
-	SHUTDOWN
-	LASTSAVE
-	DEBUG commands
-	COMMAND
-	SAVE
-	BGSAVE
-	BGREWRITEAOF

Table 1-19 Redis commands disabled in Proxy Cluster DCS Redis 3.0 instances

Keys	Server	List	Transactions	Connection	Cluster	codis
MIGRATE	SLAVEOF	BLPOP	DISCARD	SELECT	CLUSTER	TIME
MOVE	SHUTDOWN	BRPOP	EXEC	-	-	SLOTSINFO
-	LASTSAVE	BRPOPLPUSH	MULTI	-	-	SLOTSDEL
-	DEBUG commands	-	UNWATCH	-	-	SLOTSMGRTSLOT
-	COMMAND	-	WATCH	-	-	SLOTSMGRTONE
-	SAVE	-	-	-	-	SLOTSCHECK
-	BGSAVE	-	-	-	-	SLOTSMGRTTAGSLOT
-	BGREWRITEAOF	-	-	-	-	SLOTSMGRTTAGONE
-	SYNC	-	-	-	-	-
-	PSYNC	-	-	-	-	-

Keys	Server	List	Transactions	Connection	Cluster	codis
-	MONITOR	-	-	-	-	-
-	CLIENT commands	-	-	-	-	-
-	OBJECT	-	-	-	-	-
-	ROLE	-	-	-	-	-

1.5.2 Redis 4.0 Commands

DCS for Redis 4.0 is developed based on Redis 4.0.14 and is compatible with open-source protocols and commands.

This section describes DCS for Redis 4.0's compatibility with Redis commands, including supported and disabled commands. For more information about the command syntax, visit the **Redis official website**.

DCS for Redis instances support most Redis commands, which are listed in **Commands Supported by DCS for Redis 4.0**. Any client compatible with the Redis protocol can access DCS.

- For security purposes, some Redis commands are disabled in DCS, as listed in **Commands Disabled by DCS for Redis 4.0**.
- Some Redis commands are supported by cluster DCS instances for multi-key operations in the same slot. For details, see **Command Restrictions**.
- Some Redis commands have usage restrictions, which are described in **Other Command Usage Restrictions**.

Commands Supported by DCS for Redis 4.0

Table 1-20 and **Table 1-21** list the Redis commands supported by single-node, master/standby, and Redis Cluster DCS Redis 4.0 instances.

NOTE

- Commands available since later Redis versions are not supported by earlier-version instances. Run a command on redis-cli to check whether it is supported by DCS for Redis. If the message "(error) ERR unknown command" is returned, the command is not supported.
- For DCS Redis 4.0 instances in the Redis Cluster mode, ensure that all commands in a pipeline are executed on the same shard.

Table 1-20 Commands supported by single-node, master/standby, and Redis Cluster DCS Redis 4.0 instances (1)

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	KEYS
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT KILL
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYCORE	CLIENT GETNAME
RENAMENX	INCR	HSET	LREM	SPOP	ZREVRANGE	CLIENT SETNAME
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	CONFIG GET
SORT	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	MONITOR
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	SLOWLOG
TYPE	MSET	HSTRLEN	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	ROLE
SCAN	MSETNX	HLEN	RPOPLPUSH	SSCAN	ZINTERSTORE	SWAPDB
OBJECT	PSETEX	-	RPUSH	-	ZSCAN	MEMORY
PEXPIRE	SET	-	RPUSHX	-	ZRANGEBYLEX	CONFIG
PEXPIREAT	SETBIT	-	LPUSH	-	ZLEXCOUNT	COMMAND

Keys	String	Hash	List	Set	Sorted Set	Server
-	SETEX	-	-	-	ZREMRANGEBYSCORE	-
-	SETNX	-	-	-	ZREM	-
-	SETRANGE	-	-	-	-	-
-	STRLEN	-	-	-	-	-
-	BITFIELD	-	-	-	-	-

Table 1-21 Commands supported by single-node, master/standby, and Redis Cluster DCS Redis 4.0 instances (2)

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST
-	SUBSCRIBE	WATCH	SELECT (not supported by Redis Cluster instances)	SCRIPT KILL	GEORADIUS
-	UNSUBSCRIBE	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER

Commands Disabled by DCS for Redis 4.0

The following lists commands disabled by DCS for Redis 4.0.

Table 1-22 Redis commands disabled in single-node and master/standby DCS Redis 4.0 instances

Keys	Server
MIGRATE	SLAVEOF
-	SHUTDOWN
-	LASTSAVE
-	DEBUG commands
-	SAVE
-	BGSAVE
-	BGREWRITEAOF
-	SYNC
-	PSYNC

Table 1-23 Redis commands disabled in Redis Cluster DCS Redis 4.0 instances

Keys	Server	Cluster
MIGRATE	SLAVEOF	CLUSTER MEET
-	SHUTDOWN	CLUSTER FLUSHSLOTS
-	LASTSAVE	CLUSTER ADDSLOTS
-	DEBUG commands	CLUSTER DELSLOTS
-	SAVE	CLUSTER SETSLOT
-	BGSAVE	CLUSTER BUMPEPOCH
-	BGREWRITEAOF	CLUSTER SAVECONFIG
-	SYNC	CLUSTER FORGET
-	PSYNC	CLUSTER REPLICATE
-	-	CLUSTER COUNT-FAILURE-REPORTS
-	-	CLUSTER FAILOVER
-	-	CLUSTER SET-CONFIG-EPOCH
-	-	CLUSTER RESET

1.5.3 Redis 5.0 Commands

DCS for Redis 5.0 is developed based on Redis 5.0.14 and is compatible with open-source protocols and commands.

This section describes DCS for Redis 5.0's compatibility with Redis commands, including supported and disabled commands. For more information about the command syntax, visit the **Redis official website**.

DCS for Redis instances support most Redis commands. Any client compatible with the Redis protocol can access DCS.

- For security purposes, some Redis commands are disabled in DCS, as listed in **Commands Disabled by DCS for Redis 5.0**.
- Some Redis commands are supported by cluster DCS instances for multi-key operations in the same slot. For details, see **Command Restrictions**.
- Some Redis commands have usage restrictions, which are described in **Other Command Usage Restrictions**.

Commands Supported by DCS for Redis 5.0

- **Table 1-24** and **Table 1-25** list commands supported by single-node, master/standby, and Redis Cluster DCS for Redis 5.0.

NOTE

- Commands available since later Redis versions are not supported by earlier-version instances. Run a command on redis-cli to check whether it is supported by DCS for Redis. If the message "(error) ERR unknown command" is returned, the command is not supported.
- For DCS Redis 5.0 instances in the Redis Cluster mode, ensure that all commands in a pipeline are executed on the same shard.

Table 1-24 Commands supported by single-node, master/standby, and Redis Cluster DCS Redis 5.0 instances (1)

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	KEYS

Keys	String	Hash	List	Set	Sorted Set	Server
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT KILL
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYCORE	CLIENT GETNAME
RENAMENX	INCR	HSET	LREM	SPOP	ZREVRANGE	CLIENT SETNAME
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	CONFIG GET
SORT	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	MONITOR
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	SLOWLOG
TYPE	MSET	HSTRLEN	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	ROLE
SCAN	MSETNX	HLEN	RPOPLPUSH	SSCAN	ZINTERSTORE	SWAPDB
OBJECT	PSETEX	-	RPUSH	-	ZSCAN	MEMORY
PEXPIREAT	SET	-	RPUSHX	-	ZRANGEBYLEX	CONFIG
PEXPIRE	SETBIT	-	LPUSH	-	ZLEXCOUNT	COMMAND
-	SETEX	-	-	-	ZPOPMIN	-
-	SETNX	-	-	-	ZPOPMAX	-
-	SETRANGE	-	-	-	ZREMRANGEBYSCORE	-
-	STRLEN	-	-	-	ZREM	-
-	BITFIELD	-	-	-	-	-

Table 1-25 Commands supported by single-node, master/standby, and Redis Cluster DCS Redis 5.0 instances (2)

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo	Stream
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD	XACK
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH	XADD
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS	XCLAIM
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST	XDEL
-	SUBSCRIBE	WATCH	SELECT (not supported by Redis Cluster instances)	SCRIPT KILL	GEORADIUS	XGROUP
-	UNSUBSCRIBE	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER	XINFO
-	-	-	-	-	-	XLEN
-	-	-	-	-	-	XPENDING
-	-	-	-	-	-	XRANGE
-	-	-	-	-	-	XREAD
-	-	-	-	-	-	XREADGROUP
-	-	-	-	-	-	XREVRANGE
-	-	-	-	-	-	XTRIM

Commands Disabled by DCS for Redis 5.0

The following lists commands disabled by DCS for Redis 5.0.

Table 1-26 Redis commands disabled in single-node and master/standby Redis 5.0 instances

Keys	Server
MIGRATE	SLAVEOF
-	SHUTDOWN
-	LASTSAVE
-	DEBUG commands
-	SAVE
-	BGSAVE
-	BGREWRITEAOF
-	SYNC
-	PSYNC

Table 1-27 Redis commands disabled in Redis Cluster DCS Redis 5.0 instances

Keys	Server	Cluster
MIGRATE	SLAVEOF	CLUSTER MEET
-	SHUTDOWN	CLUSTER FLUSHSLOTS
-	LASTSAVE	CLUSTER ADDSLOTS
-	DEBUG commands	CLUSTER DELSLOTS
-	SAVE	CLUSTER SETSLOT
-	BGSAVE	CLUSTER BUMPEPOCH
-	BGREWRITEAOF	CLUSTER SAVECONFIG
-	SYNC	CLUSTER FORGET
-	PSYNC	CLUSTER REPLICATE
-	-	CLUSTER COUNT-FAILURE-REPORTS
-	-	CLUSTER FAILOVER
-	-	CLUSTER SET-CONFIG-EPOCH
-	-	CLUSTER RESET

1.5.4 Redis 6.0 Commands

DCS for Redis 6.0 is compatible with Redis 6.2.7 and with open-source protocols and commands.

This section describes DCS for Redis 6.0's command compatibility, including supported and disabled commands.

For more information about the command syntax, visit the **Redis official website**.

DCS Redis instances support most Redis commands. Any client compatible with the Redis protocol can access DCS.

- For security purposes, some Redis commands are disabled in DCS, as listed in **Commands Disabled by DCS for Redis 6.0**.
- Some Redis commands (such as **KEYS**, **FLUSHDB**, and **FLUSHALL**) have usage restrictions, which are described in **Other Command Usage Restrictions**.

Commands Supported by DCS for Redis 6.0

Table 1-28 Commands supported by DCS for Redis 6.0 (1)

Generic (Key)	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	CONFIG GET
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	MONITOR
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	SLOWLOG
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYCORE	ROLE
RENAMENX	INCR	HSET	LREM	SPOP	ZREVRANGE	SWAPDB

Generic (Key)	String	Hash	List	Set	Sorted Set	Server
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	MEMORY
SORT	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	CONFIG
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	ACL
TYPE	MSET	HSTRLEN	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	COMMAND
SCAN	MSETNX	HLEN	RPOPLPUSH	SSCAN	ZINTERSTORE	-
OBJECT	PSETEX	-	RPUSH	SMISMEMBER	ZSCAN	-
PEXPIREAT	SET	-	RPUSHX	-	ZRANGEBYLEX	-
PEXPIRE	SETBIT	-	LPUSH	-	ZLEXCOUNT	-
KEYS	SETEX	-	BLMOVE	-	ZPOPMIN	-
COPY	SETNX	-	LMOVE	-	ZPOPMAX	-
-	SETRANGE	-	LPOS	-	ZREMRANGEBYSCORE	-
-	STRLEN	-	-	-	ZREM	-
-	BITFIELD	-	-	-	ZDIFF	-
-	BITFIELD_RO	-	-	-	ZDIFFSTORE	-
-	GETDEL	-	-	-	ZINTER	-
-	GETEX	-	-	-	ZMSCORE	-
-	-	-	-	-	ZRANDMEMBER	-
-	-	-	-	-	ZRANGESTORE	-
-	-	-	-	-	ZUNION	-

Table 1-29 Commands supported by DCS for Redis 6.0 (2)

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo	Stream
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD	XACK
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH	XADD
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS	XCLAIM
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST	XDEL
-	SUBSCRIBE	WATCH	SELECT (not supported by Redis Cluster instances)	SCRIPT KILL	GEORADIUS	XGROUP
-	UNSUBSCRIBE	-	CLIENT CACHING	SCRIPT LOAD	GEORADIUSBYMEMBER	XINFO
-	-	-	CLIENT GETREDIR	-	-	XLEN
-	-	-	CLIENT INFO	-	-	XPENDING
-	-	-	CLIENT TRACKING	-	-	XRANGE
-	-	-	CLIENT TRACKINGINFO	-	-	XREAD
-	-	-	CLIENT UNPAUSE	-	-	XREADGROUP
-	-	-	CLIENT KILL	-	-	XREVRANGE
-	-	-	CLIENT LIST	-	-	XTRIM

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo	Stream
-	-	-	CLIENT GETNAME	-	-	XAUTOCLAIM
-	-	-	CLIENT SETNAME	-	-	XGROUP CREATECONSUMER
-	-	-	HELLO	-	-	-
-	-	-	RESET	-	-	-

Commands Disabled by DCS for Redis 6.0

Table 1-30 Redis commands disabled in DCS Redis 6.0 instances

Generic (Key)	Server	Cluster
MIGRATE	SLAVEOF	CLUSTER MEET
-	SHUTDOWN	CLUSTER FLUSHSLOTS
-	LASTSAVE	CLUSTER ADDSLOTS
-	DEBUG commands	CLUSTER DELSLOTS
-	SAVE	CLUSTER SETSLOT
-	BGSAVE	CLUSTER BUMPEPOCH
-	BGREWRITEAOF	CLUSTER SAVECONFIG
-	SYNC	CLUSTER FORGET
-	PSYNC	CLUSTER REPLICATE
-	-	CLUSTER COUNT-FAILURE-REPORTS
-	-	CLUSTER FAILOVER
-	-	CLUSTER SET-CONFIG-EPOCH
-	-	CLUSTER RESET

1.5.5 Web CLI Commands

Web CLI is a command line tool provided on the DCS console. This section describes Web CLI's compatibility with Redis commands, including supported and disabled commands. For details about the command syntax, visit the **Redis official website**.

Currently, only DCS for Redis 4.0 and later support Web CLI.

NOTE

- Keys and values cannot contain spaces.
- If the value is empty, **nil** is returned after the **GET** command is executed.

Commands Supported by Web CLI

The following lists the commands supported when you use Web CLI.

Table 1-31 Commands supported by Web CLI (1)

Keys	String	List	Set	Sorted Set	Server
DEL	APPEND	RPUSH	SADD	ZADD	FLUSHALL
OBJECT	BITCOUNT	RPUSHX	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	LLEN	SINTERSTORE	ZRANGEBYSCORE	CLIENT KILL
PTTL	GET	LPOP	SISMEMBER	ZRANK	CLIENT LIST
RANDOMKEY	GETRANGE	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT GETNAME
RENAME	GETSET	LRANGE	SMOVE	ZREMRANGEBYCORE	CLIENT SETNAME
RENAMENX	INCR	LREM	SPOP	ZREVRANGE	CONFIG GET
SCAN	INCRBY	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	SLOWLOG
SORT	INCRBYFLOAT	LTRIM	SREM	ZREVRANK	ROLE
TTL	MGET	RPOP	SUNION	ZSCORE	SWAPDB
TYPE	MSET	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	MEMORY

Keys	String	List	Set	Sorted Set	Server
-	MSETNX	RPOPLPUSH	SSCAN	ZINTERSTORE	-
-	PSETEX	-	-	ZSCAN	-
-	SET	-	-	ZRANGEBYLEX	-
-	SETBIT	-	-	ZLEXCOUNT	-
-	SETEX	-	-	-	-
-	SETNX	-	-	-	-
-	SETRANGE	-	-	-	-
-	STRLEN	-	-	-	-
-	BITFIELD	-	-	-	-

Table 1-32 Commands supported by Web CLI (2)

Hash	HyperLoglog	Connection	Scripting	Geo	Pub/Sub
HDEL	PFADD	AUTH	EVAL	GEOADD	UNSUBSCRIBE
HEXISTS	PFCOUNT	ECHO	EVALSHA	GEOHASH	PUBLISH
HGET	PFMERGE	PING	SCRIPT EXISTS	GEOPOS	PUBSUB
HGETALL	-	QUIT	SCRIPT FLUSH	GEODIST	PUNSUBSCRIBE
HINCRBY	-	-	SCRIPT KILL	GEORADIUS	-
HINCRBYFLOAT	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER	-
HKEYS	-	-	-	-	-
HMGET	-	-	-	-	-
HMSET	-	-	-	-	-
HSET	-	-	-	-	-
HSETNX	-	-	-	-	-
HVALS	-	-	-	-	-
HSCAN	-	-	-	-	-

Hash	HyperLog log	Connect ion	Scripting	Geo	Pub/Sub
HSTRLEN	-	-	-	-	-

Commands Disabled in Web CLI

The following lists the commands disabled when you use Web CLI.

Table 1-33 Commands disabled in Web CLI (1)

Keys	Server	Transactions	Cluster
MIGRATE	SLAVEOF	UNWATCH	CLUSTER MEET
WAIT	SHUTDOWN	REPLICAOF	CLUSTER FLUSHSLOTS
DUMP	DEBUG commands	DISCARD	CLUSTER ADDSLOTS
RESTORE	CONFIG SET	EXEC	CLUSTER DELSLOTS
-	CONFIG REWRITE	MULTI	CLUSTER SETSLOT
-	CONFIG RESETSTAT	WATCH	CLUSTER BUMPEPOCH
-	SAVE	-	CLUSTER SAVECONFIG
-	BGSAVE	-	CLUSTER FORGET
-	BGREWRITEAOF	-	CLUSTER REPLICATE
-	COMMAND	-	CLUSTER COUNT-FAILURE-REPORTS
-	KEYS	-	CLUSTER FAILOVER
-	MONITOR	-	CLUSTER SET-CONFIG-EPOCH
-	SYNC	-	CLUSTER RESET
-	PSYNC	-	-
-	ACL	-	-
-	MODULE	-	-

Table 1-34 Commands disabled in Web CLI (2)

List	Connection	Sorted Set	Pub/Sub
BLPOP	SELECT	BZPOPMAX	PSUBSCRIBE
BRPOP	-	BZPOPMIN	SUBSCRIBE

List	Connection	Sorted Set	Pub/Sub
BLMOVE	-	BZMPOP	-
BRPOPLPUSH	-	-	-
BLMPOP	-	-	-

1.5.6 Memcached Commands

Memcached supports the TCP-based text protocol and binary protocol. Any clients compatible with a Memcached protocol can access DCS instances.

Memcached Text Protocol

The Memcached text protocol uses ASCII text to transfer commands, which helps you compile clients and debug problems. DCS Memcached instances can even be directly connected using Telnet.

Compared with the Memcached binary protocol, the Memcached text protocol is compatible with more open-source clients, but the text protocol does not support authentication.

NOTE

Clients can use the Memcached text protocol to access DCS Memcached instances only if password-free access is enabled. Password-free access means that access to DCS Memcached instances will not be username- and password-protected, and any Memcached clients that satisfy security group rules in the same VPC can access the instances. Enabling password-free access poses security risks. Exercise caution when enabling password-free access.

Table 1-35 lists the commands supported by the Memcached text protocol and describes whether these commands are supported by DCS Memcached instances.

Table 1-35 Commands supported by the Memcached text protocol

Command	Function	Supported by DCS
add	Adding data	Yes
set	Sets data, including adding or modifying data.	Yes
replace	Replaces data.	Yes
append	Adds data after the value of the specified key.	Yes
prepend	Adds data before the value of a specified key.	Yes
cas	Checks and set data.	Yes

Command	Function	Supported by DCS
get	Queries data.	Yes
gets	Queries data details.	Yes
delete	Deletes data.	Yes
incr	Adds the specified amount to the requested counter.	Yes
decr	Removes the specified amount to the requested counter.	Yes
touch	Updates the expiration time of existing data.	Yes
quit	Closes the connection.	Yes
flush_all	Clearing DCS instance data **NOTE** The value of the delay option (if any) must be **0**.	Yes
version	Queries Memcached version information.	Yes
stats	Manages operation statistics. **NOTE** Currently, only basic statistics can be queried. Commands on optional parameters cannot be queried.	Yes
cache_memlimit	Adjusts the cache memory limit.	No
slabs	Queries usage of internal storage structures.	No
lru	Manages policies of deleting expired data.	No
lru_crawler	Manages threads of deleting expired data.	No
verbosity	Sets the verbosity level of the logging output.	No
watch	Inspects what's going on internally.	No

Memcached Binary Protocol

The Memcached binary protocol encodes commands and operations into specific structures before sending them. Commands are represented by predefined character strings.

The Memcached binary protocol provides more features but fewer clients than the Memcached text protocol. The Memcached binary protocol is more secure than the Memcached text protocol as it additionally supports simple authentication and security layer (SASL) authentication.

Table 1-36 lists the commands supported by the Memcached binary protocol and describes whether these commands are supported by DCS Memcached instances.

Table 1-36 Commands supported by the Memcached binary protocol

Command Code	Command	Function	Supported by DCS
0x00	GET	Queries data.	Yes
0x01	SET	Sets data, including adding or modifying data.	Yes
0x02	ADD	Adding data	Yes
0x03	REPLACE	Replaces data.	Yes
0x04	DELETE	Deletes data.	Yes
0x05	INCREMENT	Adds the specified amount to the requested counter.	Yes
0x06	DECREMENT	Removes the specified amount to the requested counter.	Yes
0x07	QUIT	Closes the connection.	Yes
0x08	FLUSH	Clearing DCS instance data **NOTE** The value of the delay option (if any) must be **0**.	Yes
0x09	GETQ	Queries data. The client will not receive any response in case of failure.	Yes
0x0a	NOOP	No-operation instruction, equivalent to ping.	Yes
0x0b	VERSION	Queries Memcached version information.	Yes
0x0c	GETK	Queries data and adds a key into the response packet.	Yes
0x0d	GETKQ	Queries data and returns a key. The client will not receive any response in case of failure.	Yes
0x0e	APPEND	Adds data after the value of the specified key.	Yes
0x0f	PREPEND	Adds data before the value of a specified key.	Yes

Command Code	Command	Function	Supported by DCS
0x10	STAT	Queries statistics of DCS Memcached instances. **NOTE** Currently, only basic statistics can be queried. Commands on optional parameters cannot be queried.	Yes
0x11	SETQ	Sets data, including adding or modifying data. The **SETQ** command only returns a response on failures. The client will not receive any response in the case of success.	Yes
0x12	ADDQ	Adds data. The client will not receive any response in the case of success.	Yes
0x13	REPLACEQ	Replaces data. The client will not receive any response in the case of success.	Yes
0x14	DELETEQ	Deletes data. The client will not receive any response in the case of success.	Yes
0x15	INCREMENTQ	Adds the specified amount to the requested counter. The client will not receive any response in the case of success.	Yes
0x16	DECREMENTQ	Removes the specified amount to the requested counter. The client will not receive any response in the case of success.	Yes
0x17	QUITQ	Closes the connection.	Yes
0x18	FLUSHQ	Clears data and returns no information. **NOTE** The value of the delay option (if any) must be **0**.	Yes
0x19	APPENDQ	Adds data after the value of the specified key. The client will not receive any response in the case of success.	Yes
0x1a	PREPENDQ	Adds data before the value of a specified key. The client will not receive any response in the case of success.	Yes

Command Code	Command	Function	Supported by DCS
0x1c	TOUCH	Updates the expiration time of existing data.	Yes
0x1d	GAT	Queries data and updates the expiration time of existing data.	Yes
0x1e	GATQ	Queries data and returns a key. The client will not receive any response in case of failure.	Yes
0x23	GATK	Queries data, adds a key into the response packet, and updates the expiration time of existing data.	Yes
0x24	GATKQ	Queries data, returns a key, and updates the expiration time of existing data. The client will not receive any response in case of failure.	Yes
0x20	SASL_LIST_ MECHS	Asks the server what SASL authentication mechanisms it supports.	Yes
0x21	SASL_AUTH	Starts SASL authentication.	Yes
0x22	SASL_STEP	Further authentication steps are required.	Yes

1.5.7 Command Restrictions

Some Redis commands are supported by Redis Cluster DCS instances for multi-key operations in the same slot. For details, see **Table 1-37**.

Some commands support multiple keys but do not support cross-slot access. For details, see **Table 1-38**.

Table 1-37 Redis commands restricted in Redis Cluster DCS instances

Category	Description
Set	
SINTER	Returns the members of the set resulting from the intersection of all the given sets.
SINTERSTORE	Equal to **SINTER**, but instead of returning the result set, it is stored in *destination*.
SUNION	Returns the members of the set resulting from the union of all the given sets.

Category	Description
SUNIONSTORE	Equal to **SUNION**, but instead of returning the result set, it is stored in *destination*.
SDIFF	Returns the members of the set resulting from the difference between the first set and all the successive sets.
SDIFFSTORE	Equal to **SDIFF**, but instead of returning the result set, it is stored in *destination*.
SMOVE	Moves **member** from the set at **source** to the set at *destination*.
Sorted Set	
ZUNIONSTORE	Computes the union of *numkeys* sorted sets given by the specified keys.
ZINTERSTORE	Computes the intersection of *numkeys* sorted sets given by the specified keys.
HyperLogLog	
PFCOUNT	Returns the approximated cardinality computed by the HyperLogLog data structure stored at the specified variable.
PFMERGE	Merges multiple HyperLogLog values into a unique value.
Keys	
RENAME	Renames *key* to *newkey*.
RENAMENX	Renames *key* to *newkey* if *newkey* does not yet exist.
BITOP	Performs a bitwise operation between multiple keys (containing string values) and stores the result in the destination key.
RPOPLPUSH	Returns and removes the last element (tail) of the list stored at source, and pushes the element at the first element (head) of the list stored at *destination*.
String	
MSETNX	Merges multiple HyperLogLog values into a unique value.

NOTE

While running commands that take a long time to run, such as **FLUSHALL**, DCS instances may not respond to other commands and may change to the faulty state. After the command finishes executing, the instance will return to normal.

Table 1-38 Multi-key commands of Proxy Cluster instances

Category	Command
Multi-key commands that support cross-slot access	DEL, MGET, MSET, EXISTS, SUNION, SINTER, SDIFF, SUNIONSTORE, SINTERSTORE, SDIFFSTORE, ZUNIONSTORE, ZINTERSTORE
Multi-key commands that do not support cross-slot access	SMOVE, SORT, BITOP, MSETNX, RENAME, RENAMENX, BLPOP, BRPOP, RPOPLPUSH, BRPOPLPUSH, PFMERGE, PFCOUNT, BLMOVE, COPY, GEOSEARCHSTORE, LMOVE, ZRANGESTORE

1.5.8 Other Command Usage Restrictions

This section describes restrictions on some Redis commands.

KEYS Command

In case of a large amount of cached data, running the **KEYS** command may block the execution of other commands for a long time or occupy exceptionally large memory. Therefore, when running the **KEYS** command, describe the exact pattern and do not use fuzzy **keys ***. Do not use the **KEYS** command in the production environment. Otherwise, the service running will be affected.

Commands in the Server Group

- While running commands that take a long time to run, such as **FLUSHALL**, DCS instances may not respond to other commands and may change to the faulty state. After the command finishes executing, the instance will return to normal.
- When the **FLUSHDB** or **FLUSHALL** command is run, execution of other service commands may be blocked for a long time in case of a large amount of cached data.

EVAL and EVALSHA Commands

- When the **EVAL** or **EVALSHA** command is run, at least one key must be contained in the command parameter. Otherwise, the error message "ERR eval/evalsha numkeys must be bigger than zero in redis cluster mode" is displayed.
- When the **EVAL** or **EVALSHA** command is run, a cluster DCS Redis instance uses the first key to compute slots. Ensure that the keys to be operated in your code are in the same slot. For details, visit **https://redis.io/commands**.
- For the **EVAL** command:
  - You are advised to learn the Lua script features of Redis before running the **EVAL** command. For details, see **https://redis.io/commands/eval**.
  - The execution timeout time of a Lua script is 5 seconds. Time-consuming statements such as long-time sleep and large loop statements should be avoided.

  - When calling a Lua script, do not use random functions to specify keys. Otherwise, the execution results are inconsistent on the master and standby nodes.

Debugging Lua Scripts

When you debug Lua scripts for Proxy Cluster and read/write splitting instances, only the asynchronous non-blocking mode **--ldb** is supported. The synchronous blocking mode **--ldb-sync-mode** is not supported. By default, the maximum concurrency on each proxy is **2**. This restriction does not apply to other instance types.

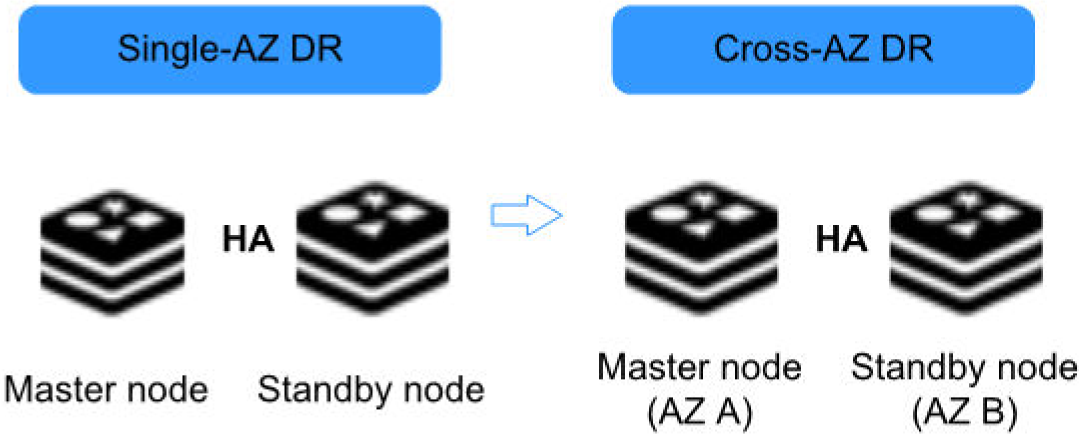
Other Restrictions

- The time limit for executing a Redis command is 15 seconds. To prevent other services from failing, a master/replica switchover will be triggered after the command execution times out.

1.6 Disaster Recovery and Multi-Active Solution

Whether you use DCS as the frontend cache or backend data store, DCS is always ready to ensure data reliability and service availability. The following figure shows the evolution of DCS DR architectures.

Figure 1-9 DCS DR architecture evolution



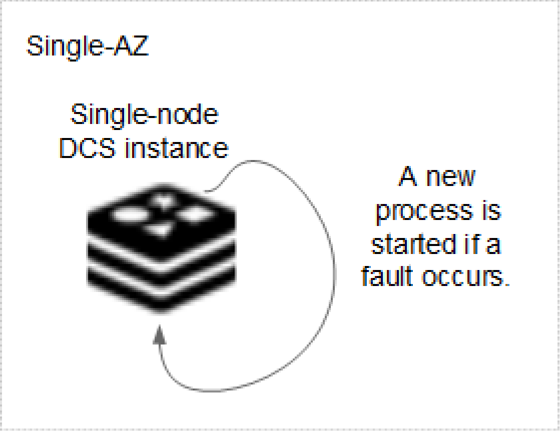
To meet the reliability requirements of your data and services, you can choose to deploy your DCS instance within a single AZ or across AZs.

Single-AZ HA

Single-AZ deployment means to deploy an instance within a physical equipment room. DCS provides process/service HA, data persistence, and hot standby DR policies for different types of DCS instances.

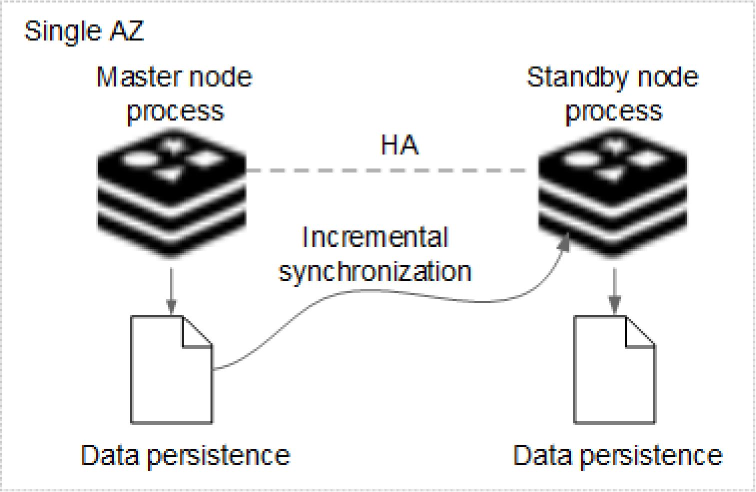
Single-node DCS instance: When DCS detects a process fault, a new process is started to ensure service HA.

Figure 1-10 HA for a single-node DCS instance deployed within an AZ



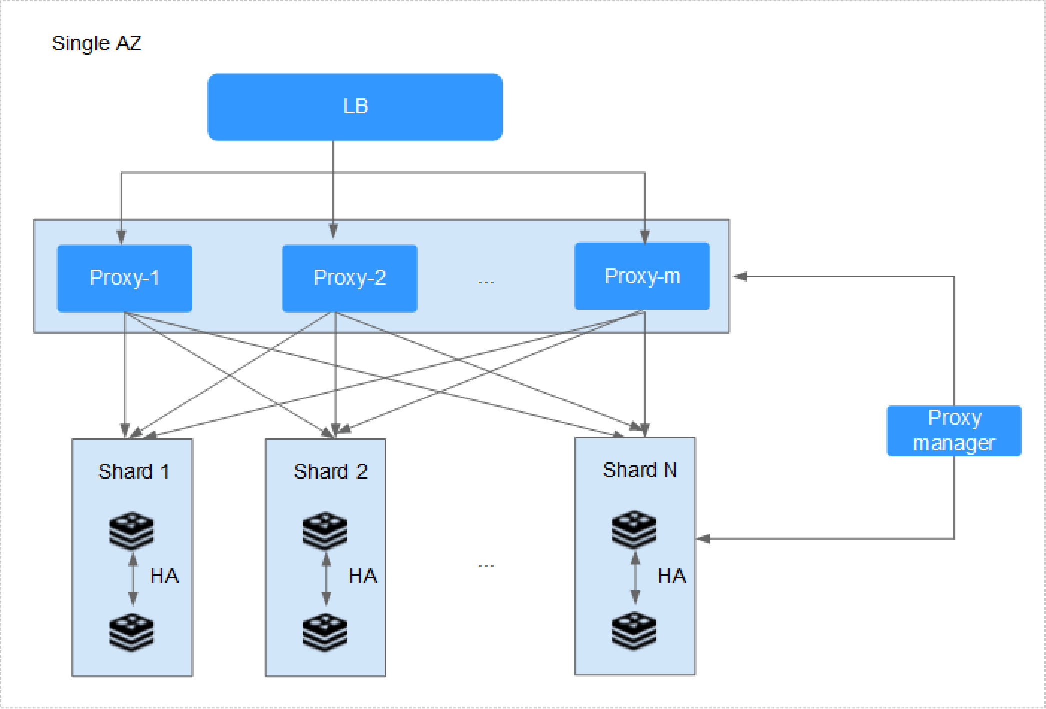
Master/Standby DCS instance: Data is persisted to disk in the master node and incrementally synchronized and persisted to the standby node, achieving hot standby and data persistence.

Figure 1-11 HA for a master/standby DCS instance deployed within an AZ



Cluster DCS instance: Similar to a master/standby instance, data in each shard (instance process) of a cluster instance is synchronized between master and standby nodes and persisted on both nodes.

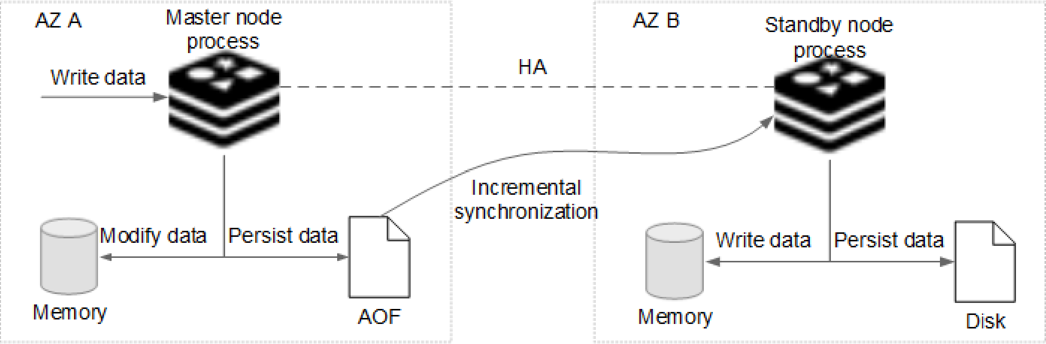
Figure 1-12 HA for a cluster DCS instance deployed within an AZ



Cross-AZ DR

The master and standby nodes of a master/standby or cluster DCS instance can be deployed across AZs (in different equipment rooms). Power supplies and networks of different AZs are physically isolated. When a fault occurs in the AZ where the master node is deployed, the standby node connects to the client and takes over data read and write operations.

Figure 1-13 Cross-AZ deployment of a master/standby DCS instance

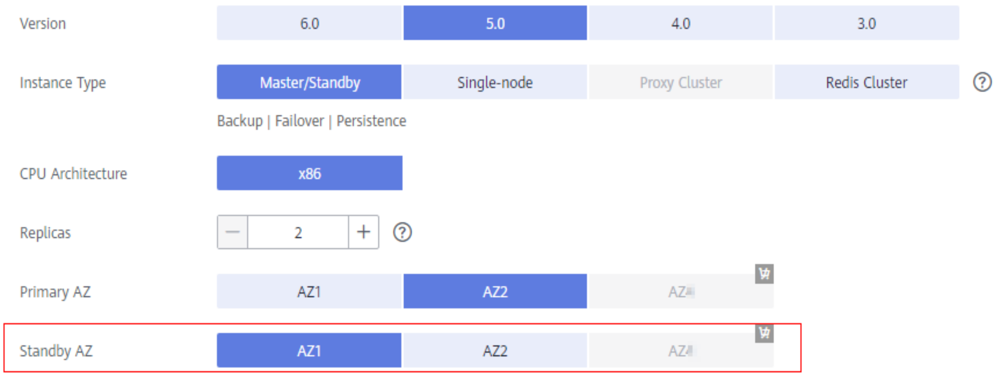


NOTE

The figure above shows the mechanism for master/standby instances. This mechanism applies in a similar way to a cluster DCS instance. Each shard (process) can be deployed across AZs.

When creating a master/standby, or cluster DCS instance, select a standby AZ that is different from the primary AZ.

Figure 1-14 Selecting different AZs



Version: 5.0

Instance Type: Master/Standby

Backup | Failover | Persistence

CPU Architecture: x86

Replicas: 2

Primary AZ: AZ2

Standby AZ: AZ1

NOTE

- You can deploy your application across AZs to ensure both data reliability and service availability in the event of power supply or network disruptions.
- Cross-AZ instances do not support password changes, command renaming, and specification modification when an AZ is faulty.

1.7 Comparing Redis Versions

When creating a DCS Redis instance, you can select the cache engine version and the instance type.

- **Version**

  DCS supports Redis 3.0/4.0/5.0/6.0. The following table describes the differences between these versions.

Table 1-39 Differences between Redis versions

Feature	Redis 3.0	Redis 4.0 & Redis 5.0	Redis 6.0
Open-source compatibility	Redis 3.0.7	Redis 4.0.14 and 5.0.14, respectively	6.2.7
Instance deployment mode	Based on VMs	Containerized based on physical servers	Containerized based on physical servers

Feature	Redis 3.0	Redis 4.0 & Redis 5.0	Redis 6.0
Time required for creating an instance	3–15 minutes, or 10–30 minutes for cluster instances.	8 seconds	8 seconds
QPS	100,000 QPS per node	100,000 QPS per node	100,000 QPS per node
Visualized data management	Not supported	Web CLI for connecting to Redis and managing data	Web CLI for connecting to Redis and managing data
Instance type	Single-node, master/standby, and Proxy Cluster	Single-node, master/standby, and Redis Cluster	Single-node, master/standby
Scale-up or scale-down	Online scale-up and scale-down	Online scale-up and scale-down	Online scale-up and scale-down
Backup and restoration	Supported for master/standby and cluster instances	Supported for master/standby and cluster instances	Supported for master/standby instances

NOTE

The underlying architectures vary by Redis version. Once a Redis version is chosen, it cannot be changed. For example, you cannot upgrade a DCS Redis 3.0 instance to Redis 4.0 or 5.0. If you require a higher Redis version, create a new instance that meets your requirements and then migrate data from the old instance to the new one.

- **Instance type**

  Select from single-node, master/standby, and cluster types. For details about their architectures and application scenarios, see **DCS Instance Types**.

1.8 Comparing Redis and Memcached

Redis and Memcached are both popular open-source in-memory databases which are easy to use and provide higher performance than relational databases.

How can I select between the two key-value databases?

Memcached is suitable for storing simple data structures, whereas Redis is suitable for storing more complex, larger data that requires persistency.

For details, see the following table.

Table 1-40 Differences between Redis and Memcached

Item	Redis	Memcached
Latency	In-memory database with sub-millisecond latency	In-memory database with sub millisecond latency
Ease of use	Simple syntax and easy to use	Simple syntax and easy to use
Distributed storage	Horizontal expansion in cluster mode	Supported
Multi-language client	Supports client connections in more than 30 languages including Java, C, and Python.	Supports client connections in more than 10 languages including Java, C, and Python.
Thread/Process	Single-core and single-thread Single-thread communication, avoiding unnecessary context switching and contention Non-blocking I/O (I/O multiplexing) is used to reduce resource consumption when multiple clients are connected.	Multi-thread and scalable The Memcached performance can be improved by increasing the number of CPUs. There is an obvious performance advantage in the scenario where the value of key is great.
Persistent storage	Supported Each write operation (adding, deleting, or modifying data) can be recorded on disk (AOF file).	Supported **NOTE** Persistence is not supported by open-source Memcached, but is supported by DCS for Memcached.
Data structure	Supports complex data structures such as hash, list, set, and sorted set, catering to various scenarios.	Supports simple strings.
Lua script support	Supported	Not supported

Item	Redis	Memcached
Snapshot backup	Supported Snapshots are generated periodically. Therefore, there is no guarantee that data will not be lost. Redis forks a subprocess to generate snapshots. When there is a large amount of data, the Redis service may be interrupted for a short time.	Not supported
Key value restriction	The value of a key can be up to 1 GB.	1 MB
Multiple databases	A single-node or master/standby DCS Redis instance supports up to 256 Redis databases. A cluster instance supports only one database, that is, DB0.	Not supported

Based on the preceding comparison, both the Redis and Memcached are easy to use and have high performance. However, Redis and Memcached are different in data structure storage, persistence, backup, migration, and script support. You are advised to select the most appropriate cache engine based on actual application scenarios.

NOTE

Memcached is suitable for caching scenarios of small amount of static data, where data is only read without further computing and processing, for example, HTML code snippets.

Redis has richer data structures and wider application scenarios.

1.9 Comparing DCS and Open-Source Cache Services

DCS supports single-node, master/standby, and cluster instances, ensuring high read/write performance and fast data access. It also supports various instance management operations to facilitate your O&M. With DCS, you only need to focus on the service logic, without concerning about the deployment, monitoring, scaling, security, and fault recovery issues.

DCS is compatible with open-source Redis and Memcached, and can be customized based on your requirements. This renders DCS unique features in addition to the advantages of open-source cache databases.

DCS for Redis vs. Open-Source Redis

Table 1-41 Differences between DCS for Redis and open-source Redis

Feature	Open-Source Redis	DCS for Redis
Service deployment	Requires 0.5 to 2 days to prepare servers.	● Creates a Redis 3.0 instance in 5 to 15 minutes. ● Creates a containerized Redis 4.0 or later instance within 8 seconds.
Version	-	Deeply engaged in the open-source community and supports the latest Redis version. Available for Redis 3.0/4.0/5.0/6.0.
Security	Network and server safety is the user's responsibility.	● Network security is ensured using VPCs and security groups. ● Data reliability is ensured by data replication and scheduled backup.
Performance	-	100,000 QPS per node
Monitoring	Provides only basic statistics.	Provides more than 30 monitoring metrics and customizable alarm threshold and policies. ● Various metrics – External metrics include the number of commands, concurrent operations, connections, clients, and denied connections. – Resource usage metrics include CPU usage, physical memory usage, network input throughput, and network output throughput. – Internal metrics include instance capacity usage, as well as the number of keys, expired keys, PubSub channels, PubSub patterns, keyspace hits, and keyspace misses. ● Customize alarm thresholds and policies for different metrics to help identify service faults.
Backup and restoration	Supported	● Supports scheduled and manual backup. Backup files can be downloaded. ● Backup data can be restored on the console.

Feature	Open-Source Redis	DCS for Redis
Parameter management	No visualized parameter management	● Visualized parameter management is supported on the console. ● Configuration parameters can be modified online. ● Data can be accessed and modified on the console.
Scale-up	Interrupts services and involves a complex procedure from modifying the server RAM to modifying Redis memory and restarting the OS and services.	● Supports online scale-up and scale-down without interrupting services. ● Specifications can be scaled up or down within the available range based on service requirements.

DCS for Memcached vs. Open-Source Memcached

Table 1-42 Differences between DCS for Memcached and open-source Memcached

Feature	Open-Source Memcached	DCS for Memcached
Service deployment	Requires 0.5 to 2 days to prepare servers.	Creates an instance in 5 to 15 minutes.
Security	Network and server safety is the user's responsibility.	● Network security is ensured using VPCs and security groups. ● Data reliability is ensured by data replication and scheduled backup.
Performance	-	100,000 QPS per node

Feature	Open-Source Memcached	DCS for Memcached
Monitoring	Provides only basic statistics.	Provides more than 30 monitoring metrics and customizable alarm threshold and policies. ● Various metrics – External metrics include the number of commands, concurrent operations, connections, clients, and denied connections. – Resource usage metrics include CPU usage, physical memory usage, network input throughput, and network output throughput. – Internal metrics include instance capacity usage, as well as the number of keys, expired keys, PubSub channels, PubSub patterns, keyspace hits, and keyspace misses. ● Customize alarm thresholds and policies for different metrics to help identify service faults.
Backup and restoration	Not supported	● Supports scheduled and manual backup. ● Backup data can be restored on the console.
Visualized maintenance	No visualized parameter management	● Visualized parameter management is supported on the console. ● Configuration parameters can be modified online.
Scale-up	Interrupts services and involves a complex procedure from modifying the server RAM to modifying Redis memory and restarting the OS and services.	● Supports online scale-up without interrupting services. ● Specifications can be scaled up or down within the available range based on service requirements.
O&M	Manual O&M	24/7 end-to-end O&M services
Data persistence	Not supported	Supported for master/standby instances

1.10 Notes and Constraints

Network

VPCs are used to manage the network security of cloud services.

- The client must be deployed on an ECS that belongs to the same VPC as the DCS instance.
- For a DCS Redis 3.0/Memcached instance, select the same security group for the DCS instance and the ECS where your client is deployed. If the DCS instance and ECS belong to different security groups, add inbound and outbound rules for the security groups. For more information on how to add the rules, see **Security Group Configurations**.
- For a DCS Redis 4.0/5.0/6.0 instance, add the IP address of the ECS where your client is deployed to the whitelist of the instance. For details, see **Managing IP Address Whitelist**.

Remarks

- You can use RESTful APIs to access DCS. Before debugging the APIs, obtain region and endpoint information from **Regions and Endpoints**.

1.11 Basic Concepts

DCS Instance

An instance is the minimum resource unit provided by DCS.

You can select the Redis or Memcached cache engine. Instance types can be single-node, master/standby, or cluster. For each instance type, multiple specifications are available.

For details, see **DCS Instance Specifications** and **DCS Instance Types**.

Project

Projects are used to group and isolate OpenStack resources (computing resources, storage resources, and network resources). A project can be a department or a project team. Multiple projects can be created for one account.

Password-Free Access

DCS Redis and Memcached instances can be accessed in the VPC without passwords. Latency is lower because no password authentication is involved.

You can enable password-free access for instances that do not have sensitive data.

For details, see **Resetting Instance Passwords**.

Cross-AZ Deployment

Master/Standby instances are deployed across different AZs with physically isolated power supplies and networks. Applications can also be deployed across AZs to achieve HA for both data and applications.

When creating a master/standby or cluster DCS Redis or Memcached instance, you can select a standby AZ for the standby node.

Shard

A shard is a management unit of a cluster DCS Redis instance. Each shard corresponds to a redis-server process. A cluster consists of multiple shards. Each shard has multiple slots. Data is distributedly stored in the slots. The use of shards increases cache capacity and concurrent connections.

Each cluster instance consists of multiple shards. By default, each shard is a master/standby instance with two replicas. The number of shards is equal to the number of master nodes in a cluster instance.

Replica

A replica is a node in a DCS instance. A single-replica instance has no standby node. A two-replica instance has one master node and one standby node. By default, each master/standby instance has two replicas. If the number of replicas is set to three for a master/standby instance, the instance has one master node and two standby nodes. A single-node instance has only one node.

1.12 Permissions

If you need to assign different permissions to employees in your enterprise to access your DCS resources, Identity and Access Management (IAM) is a good choice for fine-grained permissions management. IAM provides identity authentication, permissions management, and access control, helping you secure access to your resources.

With IAM, you can use your account to create IAM users, and assign permissions to the users to control their access to specific resources. For example, some software developers in your enterprise need to use DCS resources but should not be allowed to delete DCS instances or perform any other high-risk operations. In this scenario, you can create IAM users for the software developers and grant them only the permissions required for using DCS resources.

If your account does not require individual IAM users for permissions management, skip this section.

DCS Permissions

By default, new IAM users do not have permissions assigned. You need to add a user to one or more groups, and attach permissions policies or roles to these groups. Users inherit permissions from the groups to which they are added and can perform specified operations on cloud services based on the permissions.

DCS is a project-level service deployed and accessed in specific physical regions. To assign DCS permissions to a user group, specify the scope as region-specific projects and select regions for the permissions to take effect. If **All projects** is selected, the permissions will take effect for the user group in all region-specific projects. When accessing DCS, the users need to switch to a region where they have been authorized to use this service.

You can grant users permissions by using roles and policies.

- Roles: A type of coarse-grained authorization mechanism that defines permissions related to user responsibilities. This mechanism provides only a limited number of service-level roles for authorization. When using roles to grant permissions, you must also assign other roles on which the permissions depend to take effect. However, roles are not an ideal choice for fine-grained authorization and secure access control.
- Policies: A type of fine-grained authorization mechanism that defines permissions required to perform operations on specific cloud resources under certain conditions. This mechanism allows for more flexible policy-based authorization, meeting requirements for secure access control. For example, you can grant DCS users only the permissions for operating DCS instances. Fine-grained policies are based on APIs. The minimum granularity of a policy is API actions. For the API actions supported by DCS, see section "Permissions Policies and Supported Actions" in the *Distributed Cache Service API Reference*.

Table 1-43 lists all the system permissions supported by DCS.

Table 1-43 System-defined roles and policies supported by DCS

Role/Policy Name	Description	Type	Dependency
DCS FullAccess	All permissions for DCS. Users granted these permissions can operate and use all DCS instances.	System-defined policy	None
DCS UserAccess	Common user permissions for DCS, excluding permissions for creating, modifying, deleting DCS instances and modifying instance specifications.	System-defined policy	None
DCS ReadOnlyAccess	Read-only permissions for DCS. Users granted these permissions can only view DCS instance data.	System-defined policy	None

Role/Policy Name	Description	Type	Dependency
DCS AgencyAccess	Permissions to assign to DCS agencies. These permissions are used by a tenant to delegate DCS to perform the following operations on their resources when necessary. This policy is irrelevant to the operations performed by authorized users. ● Querying a subnet ● Querying the subnet list ● Querying a port ● Querying the port list ● Updating a port ● Creating a port	System-defined policy	None

NOTE

The **DCS UserAccess** policy is different from the **DCS FullAccess** policy. If you configure both of them, you cannot create, modify, delete, or scale DCS instances because deny statements will take precedence over allowed statements.

Table 1-44 lists the common operations supported by system-defined policies for DCS.

Table 1-44 Common operations supported by each system policy

Operation	DCS FullAccess	DCS UserAccess	DCS ReadOnlyAccess
Modifying instance configuration parameters	√	√	×
Deleting background tasks	√	√	×
Accessing instances using Web CLI	√	√	×
Modifying instance running status	√	√	×

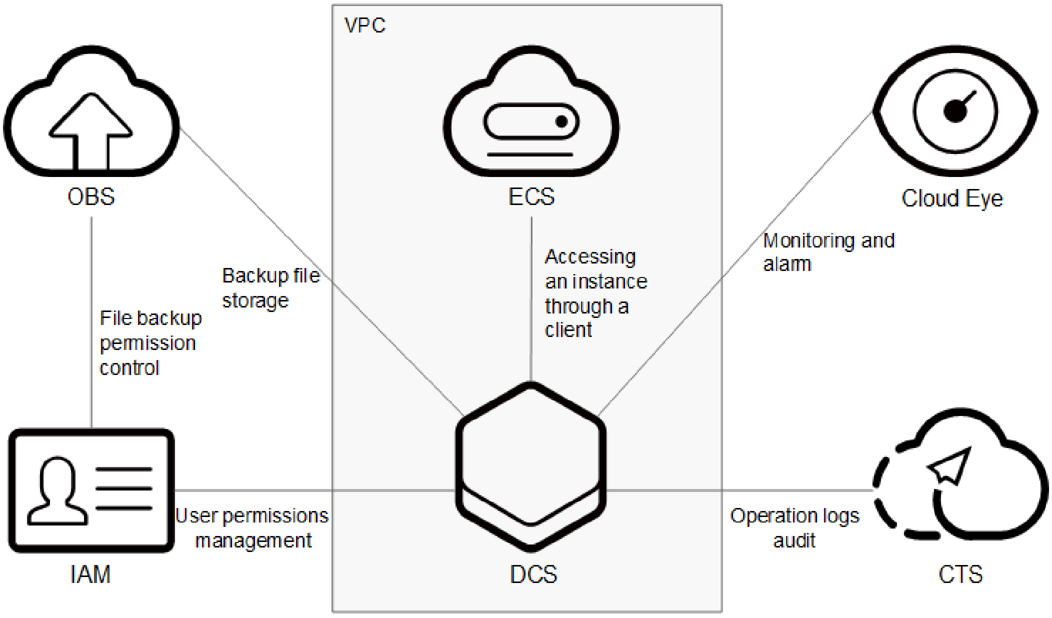
Operation	DCS FullAccess	DCS UserAccess	DCS ReadOnlyAccess
Expanding instance capacity	√	×	×
Changing instance passwords	√	√	×
Modifying DCS instances	√	×	×
Performing a master/ standby switchover	√	√	×
Backing up instance data	√	√	×
Analyzing big keys or hot keys	√	√	×
Creating DCS instances	√	×	×
Deleting instance backup files	√	√	×
Restoring instance data	√	√	×
Resetting instance passwords	√	√	×
Migrating instance data	√	√	×
Downloading instance backup data	√	√	×
Deleting DCS instances	√	×	×
Querying instance configuration parameters	√	√	√

Operation	DCS FullAccess	DCS UserAccess	DCS ReadOnlyAccess
Querying instance restoration logs	√	√	√
Querying instance backup logs	√	√	√
Querying DCS instances	√	√	√
Querying instance background tasks	√	√	√
Querying all instances	√	√	√
Viewing instance performance metrics	√	√	√
Modifying parameters in a parameter template	√	√	×
Deleting a parameter template	√	√	×
Creating a parameter template	√	√	×
Parameter template list	√	√	√
Querying a parameter template	√	√	√

1.13 Related Services

DCS is used together with other services, including VPC, ECS, IAM, Cloud Eye, CTS, and Object Storage Service (OBS).

Figure 1-15 Relationships between DCS and other services


VPC
OBS
ECS
Cloud Eye
Backup file storage
File backup permission control
Accessing an instance through a client
Monitoring and alarm
User permissions management
Operation logs audit
IAM
DCS
CTS

VPC

A VPC is an isolated virtual network environment on the cloud. You can configure IP address ranges, subnets, and security groups in a VPC.

DCS runs in VPCs. The VPC service manages EIPs and bandwidth, and provides security groups. You can configure access rules for security groups to secure the access to DCS.

ECS

An ECS is a cloud server that provides scalable, on-demand computing resources for secure, flexible, and efficient applications.

You can access and manage your DCS instances using an ECS.

IAM

IAM provides identity authentication, permissions management, and access control.

With IAM, you can control access to DCS.

Cloud Eye

Cloud Eye is a secure, scalable, and integrated monitoring service. With Cloud Eye, you can monitor your DCS service and configure alarm rules and notifications.

Cloud Trace Service (CTS)

CTS provides you with a history of operations performed on cloud service resources. With CTS, you can query, audit, and backtrack operations. The traces include the operation requests sent using the management console or open APIs and the results of these requests.

OBS

OBS provides secure, cost-effective storage service using objects as storage units. With OBS, you can store and manage the lifecycle of massive amounts of data.

You can store DCS instance backup files in OBS.

2 Permissions Management

2.1 Creating a User and Granting DCS Permissions

This chapter describes how to use IAM to implement fine-grained permissions control for your DCS resources. With IAM, you can:

- Create IAM users for employees based on your enterprise's organizational structure. Each IAM user will have their own security credentials for accessing DCS resources.
- Grant only the permissions required for users to perform a specific task.
- Entrust an account or cloud service to perform efficient O&M on your DCS resources.

If your account does not need individual IAM users, you may skip over this chapter.

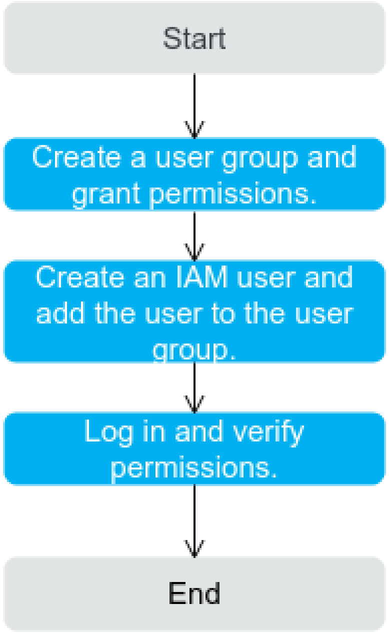
This section describes the procedure for granting the **DCS ReadOnlyAccess** permission (see **Figure 2-1**) as an example.

Prerequisites

You are familiar with the permissions (see **Permissions**) supported by DCS and choose policies or roles according to your requirements. For the permissions of other services, see **System-defined Permissions**.

Process Flow

Figure 2-1 Process of granting DCS permissions


Start
Create a user group and grant permissions.
Create an IAM user and add the user to the user group.
Log in and verify permissions.
End

1. **Create a user group and assign permissions** to it.

   Create a user group on the IAM console, and attach the **DCS ReadOnlyAccess** policy to the group.

2. **Create an IAM user**.

   Create a user on the IAM console and add the user to the group created in **1**.

3. **Log in** and verify permissions.

   Log in to the DCS console by using the newly created user, and verify that the user only has read permissions for DCS.

2.2 DCS Custom Policies

Custom policies can be created to supplement the system-defined policies of DCS. For the actions that can be added to custom policies, see section "Permissions Policies and Supported Actions" in the *Distributed Cache Service API Reference*.

You can create custom policies in either of the following ways:

- Visual editor: Select cloud services, actions, resources, and request conditions. This does not require knowledge of policy syntax.
- JSON: Edit JSON policies from scratch or based on an existing policy.

For details, see **Creating a Custom Policy**. The following section contains examples of common DCS custom policies.

NOTE

Due to data caching, a policy involving OBS actions will take effect five minutes after it is attached to a user, user group, or project.

Example Custom Policies

- Example 1: Allowing users to delete and restart DCS instances and clear data of an instance

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "
                    dcs:instance:delete
                    dcs:instance:modifyStatus
                 "
            ]
        }
    ]
}
```

- Example 2: Denying DCS instance deletion

  A policy with only "Deny" permissions must be used in conjunction with other policies to take effect. If the permissions assigned to a user contain both "Allow" and "Deny", the "Deny" permissions take precedence over the "Allow" permissions.

  The following method can be used if you need to assign permissions of the **DCS FullAccess** policy to a user but you want to prevent the user from deleting DCS instances. Create a custom policy for denying DCS instance deletion, and attach both policies to the group to which the user belongs. Then, the user can perform all operations on DCS instances except deleting DCS instances. The following is an example of a deny policy:

```
{
        "Version": "1.1",
        "Statement": [
                {
                        "Effect": "Deny",
                        "Action": [
                                "dcs:instance:delete"
                        ]
                }
        ]
}
```

3 Process of Using DCS

How to Manage DCS Instances

You can access Distributed Cache Service (DCS) from the web-based management console or by using RESTful application programming interfaces (APIs) through HTTPS requests.

- Using the console

  You can sign up and log in to the console, click ☰ in the upper left corner on the homepage, and choose **Middleware** > **Distributed Cache Service for Redis**.

  For details on how to use the DCS console, see chapters in this document.

  DCS monitoring data is recorded by Cloud Eye. To view the monitoring metrics or configure alarm rules, go to the Cloud Eye console. For details, see **Viewing DCS Monitoring Metrics**.

  If you have enabled Cloud Trace Service (CTS), DCS instance operations are recorded by CTS. You can view the operations history on the CTS console. For details, see **Viewing Traces on the CTS Console**.

- Using APIs

  DCS provides RESTful APIs for you to integrate DCS into your own application system. For details about DCS APIs and API calling, see the *Distributed Cache Service API Reference*.

NOTICE

1. For DCS instance functions with open APIs, manage them using the console or calling the APIs. For those without open APIs, manage them using the console.
2. For details about APIs for monitoring and auditing, see the Cloud Eye and Cloud Trace Service (CTS) documentation.

Using DCS

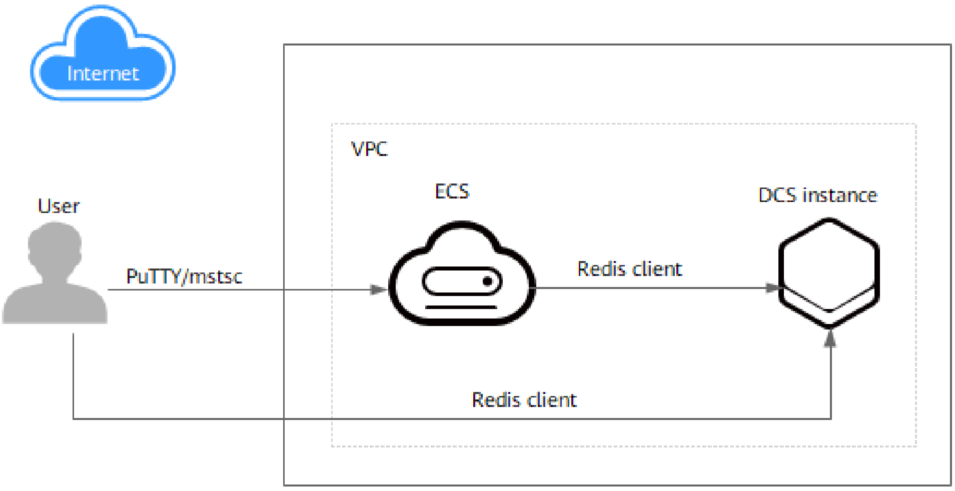
After creating a DCS instance, access it by referring to **Accessing an Instance**. Any client that is compatible with the open-source Redis or Memcached protocol can respectively access a DCS Redis or Memcached instance. After accessing a DCS instance, you can enjoy the fast read/write operations enabled by DCS.

NOTICE

DCS does not involve sensitive user information. Which, why, when, and how data is processed with DCS must comply with local laws and regulations. If sensitive data needs to be transmitted or stored, encrypt data before transmission or storage.

For details on how to access a DCS instance, see the following figure.

Figure 3-1 Accessing a DCS instance



NOTE

- The Elastic Cloud Server (ECS) where the client is and the DCS instance to be accessed should be within one Virtual Private Cloud (VPC).

4 Getting Started

4.1 Creating an Instance

4.1.1 Identifying Requirements

Before creating a DCS instance, identify your requirements and complete the following preparations:

1. Decide on the required cache engine.

   Choose a cache engine based on service requirements. The cache engine cannot be changed once the instance is created.

   – For more information about Redis and Memcached cache engines, see **What Is DCS?**

   – For more information about the differences between Redis and Memcached, see **Comparing Redis and Memcached**.

2. Decide on the required cache engine version.

   Different Redis versions have different features. For details, see **Comparing Redis Versions**.

3. Decide on the required instance type.

   DCS provides single-node, master/standby, Proxy Cluster, and Redis Cluster types of instances. Each type has its own architecture. For details about the instance architectures, see **DCS Instance Types**.

4. Decide on the required instance specification.

   Each specification specifies the maximum available memory, number of connections, and bandwidth. For details, see **DCS Instance Specifications**.

5. Decide on the region and whether cross-AZ deployment is required.

   Choose a region closest to your application to reduce latency.

   A region consists of multiple availability zones (AZs) with physically isolated power supplies and networks. Master/Standby and cluster DCS instances can be deployed across AZs.

NOTE

- If a master/standby or cluster DCS instance is deployed across AZs, faults in an AZ do not affect cache nodes in other AZs. This is because when the master node is faulty, the standby cache node will automatically become the master node to provide services. Such deployment achieves better disaster recovery.
- Deploying a DCS instance across AZs slightly reduces network efficiency compared with deploying an instance within an AZ. Therefore, if a DCS instance is deployed across AZs, synchronization between master and standby cache nodes is slightly less efficient.

6. Decide whether backup policies are required.

   Currently, backup restoration policies can be configured for instances other than single-node ones. For details about backup and restoration, see **Overview**.

4.1.2 Preparing Required Resources

To access DCS instances through a Virtual Private Cloud (VPC), create a VPC and configure security groups and subnets for it before using DCS. A VPC provides an isolated virtual network environment which you can configure and manage. Using VPCs enhances cloud resource security and simplifies network deployment.

Once you have created the required resources, you can use them for all DCS instances you subsequently create.

Creating a VPC and Subnet

Step 1 Log in to the management console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 Click **Service List**, and choose **Network** > **Virtual Private Cloud** to launch the VPC console.

Step 4 Click **Apply for VPC**.

Step 5 Create a VPC as prompted, retaining the default values unless otherwise required.

For details about how to create a VPC, see "VPC and Subnet" > "VPC" > "Creating a VPC" in *Virtual Private Cloud User Guide*.

After a VPC is created, a subnet is also created in the subnet. If the VPC needs more subnets, go to **Step 6** and **Step 7**. Otherwise, go to **Creating a Security Group**.

NOTE

- When creating a VPC, **CIDR Block** indicates the IP address range of the VPC. If this parameter is set, the IP addresses of subnets in the VPC must be within the IP address range of the VPC.
- If you create a VPC to provision DCS instances, you do not need to configure the CIDR block for the VPC.

Step 6 In the navigation pane on the left, choose **Virtual Private Cloud** > **Subnets**.

Step 7 Click **Create Subnet**. Create a subnet as prompted, retaining the default values unless otherwise required.

For details about how to create a subnet, see "VPC and Subnet" > "Subnet" in *Virtual Private Cloud User Guide*.

----End

Creating a Security Group

NOTE

Only DCS Redis 3.0 and Memcached instances require security groups.

Step 1 Log in to the VPC console.

Step 2 In the navigation pane on the left, choose **Access Control** > **Security Groups** and then click **Create Security Group** in the upper right corner of the displayed page. Create a security group as prompted, retaining the default values unless otherwise required.

For details about how to create a security group, see "Security" > "Security Group" > "Creating a Security Group" in *Virtual Private Cloud User Guide*.

- Set **Template** to **Custom**.
- After a security group is created, retain the default inbound rule that allows communication among ECSs within the security group and the default outbound rule that allows all outbound traffic.
- To use DCS, you must add the security group rules described in the following table. You can also add other rules based on site requirements.

Table 4-1 Security group rules

Direction	Protocol	Port	Source	Description
Inbound	TCP	6379	0.0.0.0/0	Access a DCS Redis 3.0 instance in a private network.
Inbound	TCP	11211	0.0.0.0/0	Access a DCS Memcached instance.

----End

4.1.3 Creating a DCS Redis Instance

You can create one or more DCS Redis instances with the required computing capabilities and storage space based on service requirements.

Prerequisites

You have prepared **necessary resources**.

Creating a DCS Redis Instance

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner of the management console and select a region and a project.

Step 3 Click **Create DCS Instance**.

Step 4 Select a region closest to your application to reduce latency and accelerate access.

Step 5 Specify the following instance parameters based on the information collected in **Identifying Requirements**.

1. **Cache Engine**:

   Select **Redis**.

2. **Version**:

   Currently, versions 3.0/4.0/5.0/6.0 are supported.

   NOTE

   – When creating a Proxy Cluster instance, you can select version 3.0.

   – When creating a Redis Cluster instance, you can select versions 4.0/5.0.

   – The Redis version cannot be changed once the instance is created. To use a later Redis version, create another DCS Redis instance and then migrate data from the old instance to the new one.

   – The method of connecting a client to a Redis Cluster instance is different from that of connecting a client to other types of instances. For details, see **Accessing a DCS Redis Instance Through redis-cli**.

3. Set **Instance Type** to **Single-node**, **Master/Standby**, **Proxy Cluster** or **Redis Cluster**.

4. Set **CPU Architecture** to **x86**.

5. Set **Replicas**. The default value is **2** (including the master).

   This parameter is displayed only when you select Redis 4.0/5.0/6.0 and the instance type is master/standby or Redis Cluster.

6. Select an AZ.

   If the instance type is master/standby, Proxy Cluster, or Redis Cluster, **Standby AZ** is displayed. Select a standby AZ for the standby node of the instance.

   NOTE

   – To accelerate access, deploy your instance and your application in the same AZ.

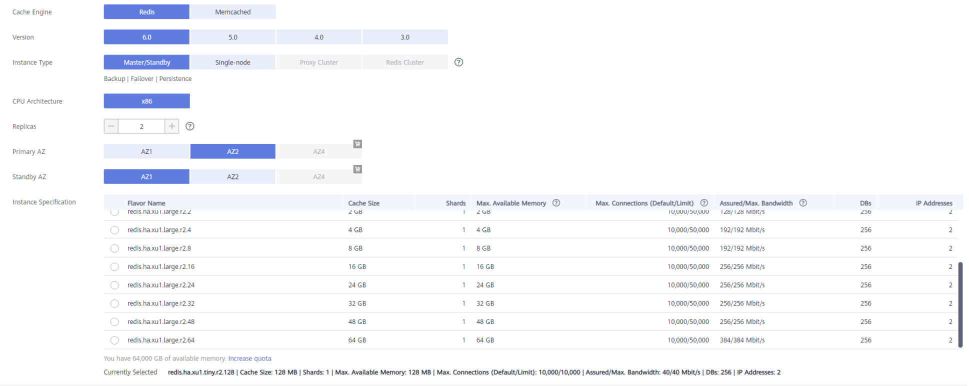
   – There are multiple AZs in each region. If resources are insufficient in an AZ, the AZ will be unavailable. In this case, select another AZ.

7. **Instance Specification**:

   The remaining quota is displayed on the console.

   To apply to increase quota, click **Increase quota** below the specifications.

Figure 4-1 Creating a DCS Redis instance



Cache Engine: **Redis** | Memcached

Version: **6.0** | 5.0 | 4.0 | 3.0

Instance Type: **Master/Standby** | Single-node | Proxy Cluster | Redis Cluster

Backup | Failover | Persistence

CPU Architecture: **x86**

Replicas: 2

Primary AZ: AZ1 | **AZ2** | AZ4

Standby AZ: **AZ1** | AZ2 | AZ4

Instance Specification

	Flavor Name	Cache Size	Shards	Max. Available Memory	Max. Connections (Default/Limit)	Assured/Max. Bandwidth	DBs	IP Addresses
○	redis.ha.xu1.large.r2.2	2 GB	1	2 GB	10,000/50,000	128/128 Mbit/s	256	2
○	redis.ha.xu1.large.r2.4	4 GB	1	4 GB	10,000/50,000	192/192 Mbit/s	256	2
○	redis.ha.xu1.large.r2.8	8 GB	1	8 GB	10,000/50,000	192/192 Mbit/s	256	2
○	redis.ha.xu1.large.r2.16	16 GB	1	16 GB	10,000/50,000	256/256 Mbit/s	256	2
○	redis.ha.xu1.large.r2.24	24 GB	1	24 GB	10,000/50,000	256/256 Mbit/s	256	2
○	redis.ha.xu1.large.r2.32	32 GB	1	32 GB	10,000/50,000	256/256 Mbit/s	256	2
○	redis.ha.xu1.large.r2.48	48 GB	1	48 GB	10,000/50,000	256/256 Mbit/s	256	2
○	redis.ha.xu1.large.r2.64	64 GB	1	64 GB	10,000/50,000	384/384 Mbit/s	256	2

You have 64,000 GB of available memory. Increase quota

Currently Selected redis.ha.xu1.tiny.r2.128 | Cache Size: 128 MB | Shards: 1 | Max. Available Memory: 128 MB | Max. Connections (Default/Limit): 10,000/10,000 | Assured/Max. Bandwidth: 40/40 Mbit/s | DBs: 256 | IP Addresses: 2

Step 6 Configure the instance network parameters.

1. Select a VPC and a subnet.
2. Configure the IP address.

   Redis Cluster instances only support automatically-assigned IP addresses. The other instance types support both automatically-assigned IP addresses and manually-specified IP addresses. You can manually specify an IP address available in your subnet.

   For a DCS Redis 4.0 or later instance, you can specify a port number in the range from 1 to 65535. If no port is specified, the default port 6379 will be used. For a DCS Redis 3.0 instance, the port cannot be customized. Port 6379 will be used.
3. Select a security group.

   A security group is a set of rules that control access to ECSs. It provides access policies for mutually trusted ECSs with the same security protection requirements in the same VPC.

   This parameter is displayed only for DCS Redis 3.0 instances. DCS Redis 4.0/5.0/6.0 instances are based on VPC endpoints and do not support security groups. To control access to a DCS Redis 4.0/5.0/6.0 instance, configure a whitelist after instance creation. For details, see **Managing IP Address Whitelist**.

Step 7 Set the instance password.

- Select **Yes** or **No** for **Password Protected**.

  NOTE

  – Password-free access carries security risks. Exercise caution when selecting this mode.

  – After creating a DCS Redis instance to be accessed in password-free mode, you can set a password for it by using the password reset function. For details, see **Changing Password Settings for DCS Redis Instances**.
- **Password** and **Confirm Password**: These parameters indicate the password of accessing the DCS Redis instance, and are displayed only when **Password Protected** is set to **Yes**.

NOTE

For security purposes, if password-free access is disabled, the system prompts you to enter an instance-specific password when you are accessing the DCS Redis instance. Keep your instance password secure and change it periodically.

Step 8 Configure **Parameter Configuration**.

You can select **Default Templates** or **Use custom template**.

NOTE

- On the instance creation page, the default parameter templates are used by default.
- If you use a custom template, the selected cache engine version and instance type must match those of the template. For details about using custom templates, see **Creating a Custom Parameter Template**.

Step 9 Choose whether to enable **Auto Backup**.

This parameter is displayed only when the instance type is master/standby or cluster. For more information on how to configure a backup policy, see **Overview**.

Step 10 Specify the number of instances to create.

Step 11 Enter an instance name and select an enterprise project.

The value of **Name** contains at least 4 characters. When you create multiple instances at a time, the instances are named in the format of *custom name*-*n*, where *n* starts from 000 and is incremented by 1. For example, if you create two instances and set **name** to **dcs_demo**, the two instances are respectively named as **dcs_demo-000** and **dcs_demo-001**.

Step 12 Click **More Settings** to configure more parameters.

1. Enter a description of the instance.
2. Rename critical commands.

   Command Renaming is displayed for Redis 4.0 and later. Currently, you can only rename the **COMMAND**, **KEYS**, **FLUSHDB**, **FLUSHALL**, **HGETALL**, **SCAN**, **HSCAN**, **SSCAN**, and **ZSCAN** commands.
3. Specify tags.

   Tags are used to identify cloud resources. When you have many cloud resources of the same type, you can use tags to classify cloud resources by dimension (for example, by usage, owner, or environment).

   If tag policies for DCS have been set in your organization, add tags to DCS instances based on these policies. If a tag does not comply with the tag policies, DCS instance creation may fail. Contact your organization administrator to learn more about tag policies.

   – If you have created predefined tags, select a predefined pair of tag key and value. Click **View predefined tags**. On the Tag Management Service (TMS) console, view predefined tags or create new tags.

   – You can also add a tag by entering the tag key and value. For details about tag naming requirements, see **Managing Tags**.

Step 13 Click **Create Now**.

The displayed page shows the instance information you have specified.

Step 14 Confirm the instance information and click **Submit**.

Step 15 Return to the **Cache Manager** page to view and manage your DCS instances.

1. Creating a single-node or master/standby DCS Redis 3.0 instance takes 5 to 15 minutes. Creating a cluster DCS Redis 3.0 instance takes 30 minutes.DCS Redis 4.0 and later instances are containerized and can be created within seconds.
2. After a DCS instance has been successfully created, it enters the **Running** state by default.

----End

4.1.4 Creating a DCS Memcached Instance

You can create one or more DCS Memcached instances with the required computing capabilities and storage space based on service requirements.

Creating a DCS Memcached Instance

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click **Create DCS Instance**.

Step 5 Select a region closest to your application to reduce latency and accelerate access.

Step 6 Specify the following instance parameters based on the information collected in **Identifying Requirements**.

1. **Cache Engine**:

   Select **Memcached**.
2. **Instance Type**:

   Select **Single-node** or **Master/Standby**.
3. Select an AZ.

   NOTE

   To accelerate access, deploy your instance and your application in the same AZ. To ensure data reliability, deploy them in different AZs.

   If the instance type is master/standby, **Standby AZ** is displayed. Select a standby AZ for the standby node of the instance.
4. Specify **Instance Specification**.

   The remaining quota is displayed on the console.

   To apply to increase quota, click **Increase quota** below the specifications.

Step 7 Configure the instance network parameters.

1. For **VPC**, select a created VPC, subnet, and specify the IP address.

   You can choose to obtain an automatically assigned IP address or manually specify an IP address that is available in the selected subnet.

2. Select a security group.

   A security group is a set of rules that control access to ECSs. It provides access policies for mutually trusted ECSs with the same security protection requirements in the same VPC.

Step 8 Set the instance password.

- Select **Yes** or **No** for **Password Protected**.

  NOTE

  – Password-free access carries security risks. Exercise caution when selecting this mode.

  – After the instance is created, you can click reset its password. For details, see **Changing Password Settings for DCS Memcached Instances**.

  – If password-free access is disabled, DCS Memcached instances must be accessed using the Memcached binary protocol and through SASL authentication.

- Username required for accessing the new DCS instance. The username must meet the following requirements.

  NOTE

  This parameter is displayed only if password-protected access is enabled.

  – Cannot be left blank.

  – Must start with a letter.

  – Can contain 1 to 64 characters.

  – Must contain only letters, digits, hyphens (-), and underscores (_).

- **Password** and **Confirm Password**: These parameters indicate the password of accessing the DCS Memcached instance, and are displayed only when **Password Protected** is set to **Yes**.

  NOTE

  For security purposes, if password-free access is disabled, the system prompts you to enter an instance-specific password when you are accessing the DCS Memcached instance. Keep your instance password secure and change it periodically.

Step 9 Choose whether to enable **Auto Backup**.

This parameter is displayed only when the instance type is master/standby. For more information on how to configure a backup policy, see **Backing Up and Restoring DCS Instances**.

Step 10 Enter the number of instances to be created.

Step 11 Enter an instance name and select an enterprise project.

The instance name can contain 4 to 64 characters.

Step 12 Click **More Settings** to configure more parameters.

1. Enter a description of the instance.
2. Specify tags.

   Tags are used to identify cloud resources. When you have many cloud resources of the same type, you can use tags to classify cloud resources by dimension (for example, by usage, owner, or environment).

Step 13 Click **Next**.

The displayed page shows the instance information you have specified.

Step 14 Confirm the instance information.

Step 15 After the new DCS instance has been created, return to the **Cache Manager** page to view and manage your DCS instances.

1. It takes 5 to 15 minutes to create a DCS instance.
2. After a DCS instance has been successfully created, it enters the **Running** state by default.

----End

4.2 Accessing an Instance

4.2.1 Network Conditions for Accessing DCS Redis

You can access a DCS instance through any Redis client. For details about Redis clients, see the **Redis official website**.

There are different constraints when a client connects to Redis in certain ways:

- Accessing a Redis instance on a client within the same VPC

  The ECS where the client is installed must be in the same VPC as the DCS Redis instance. An ECS and a DCS instance can communicate with each other only when they belong to the same VPC. **Redis 3.0:** The instance and the ECS must either be configured with the same security group or use different security groups but can communicate with each other as configured by the security group rules. **Redis 4.0/5.0/6.0/6.0 basic:** The IP address of the ECS must be on the whitelist of the DCS instance.

- Accessing a Redis instance on a client across VPCs in the same region

  If the client and DCS Redis instance are not in the same VPC, connect them by establishing a VPC peering connection. For details, see **Does DCS Support Cross-VPC Access?**

4.2.2 Accessing a DCS Redis Instance Through redis-cli

Access a DCS Redis instance through redis-cli on an ECS in the same VPC. For more information on how to use other Redis clients, visit **https://redis.io/clients**.

NOTE

- Redis 3.0 does not support port customization and allows only port 6379. For Redis 4.0 and later, you can specify a port or use the default port 6379. The following uses the default port 6379. If you have specified a port, replace 6379 with the actual port.
- **When connecting to a Redis Cluster instance, ensure that -c is added to the command.** Otherwise, the connection will fail.
  - Run the following command to connect to a Redis Cluster instance:

    ./redis-cli -h *{dcs_instance_address}* -p *6379* -a *{password}* **-c**
  - Run the following command to connect to a single-node, master/standby, or Proxy Cluster instance:

    ./redis-cli -h {dcs_instance_address} -p 6379 -a *{password}*

  For details, see **Step 3** and **Step 4**.
- If SSL is enabled for a single-node or master/standby DCS Redis 6.0 instance, set an SSL certificate path.

  ./redis-cli -h {dcs_instance_address} -p 6379 -a *{password}* **--tls --cacert *{certification file path}***
- To connect to Redis with SSL encryption, use redis-cli 6.x or later.

Prerequisites

- The DCS Redis instance you want to access is in the **Running** state.
- An ECS has been created. For more information on how to create ECSs, see the *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the GCC compilation environment has been installed on the ECS.

Procedure (Linux)

Step 1 View the IP address and port number of the DCS Redis instance to be accessed.

For details, see **Viewing Details of a DCS Instance**.

Step 2 Install redis-cli.

The following steps assume that your client is installed on the Linux OS.

1. Log in to the ECS.
2. Run the following command to download the source code package of your Redis client from **https://download.redis.io/releases/redis-6.2.13.tar.gz**:

   wget http://download.redis.io/releases/redis-6.2.13.tar.gz

   NOTE

   The following uses redis-6.2.13 as an example. For details, see the **Redis official website**.
3. Run the following command to decompress the source code package of your Redis client:

   tar -xzf redis-6.2.13.tar.gz
4. Run the following commands to go to the Redis directory and compile the source code of your Redis client:

   cd redis-6.2.13

   make

cd src

Step 3 Access a DCS instance of a type other than Redis Cluster.

Perform the following procedure to access a single-node, master/standby instance.

./redis-cli -h $*{dcs_instance_address}* -p 6379 -a $*{password}*

NOTE

1. If the instance is password-free, connect it by running the **./redis-cli -h** *${dcs_instance_address}* **-p 6379** command.
2. If the instance is password-protected, connect it by running the **./redis-cli -h** *${dcs_instance_address}* **-p 6379 -a** *${password}* command.
3. If you forget the instance password or need to reset the password, refer to **Resetting Instance Passwords**.

Step 4 Access a DCS instance of the Redis Cluster type.

Do as follows to access a Redis Cluster instance:

1. Run the following commands to access the chosen DCS Redis instance:

   ./redis-cli -h *{dcs_instance_address}* -p 6379 -a *{password}* -c

   {dcs_instance_address} indicates the IP address of the DCS Redis instance, **6379** is the port used for accessing the instance, *{password}* is the password of the instance, and **-c** is used for accessing Redis Cluster nodes. The IP address and port number are obtained in **Step 1**.

   Example:

```
root@ecs-redis:~/redis-6.2.13/src# ./redis-cli -h 192.168.0.85 -p 6379 -a ****** -c
192.168.0.85:6379>
```

2. Run the following command to view the Redis Cluster node information:

   cluster nodes

   Each shard in a Redis Cluster has a master and a replica by default. The proceeding command provides all the information of cluster nodes.

```
192.168.0.85:6379> cluster nodes
0988ae8fd3686074c9afdcce73d7878c81a33ddc 192.168.0.231:6379@16379 slave
f0141816260ca5029c56333095f015c7a058f113 0 1568084030
000 3 connected
1a32d809c0b743bd83b5e1c277d5d201d0140b75 192.168.0.85:6379@16379 myself,master - 0
1568084030000 2 connected 5461-10922
c8ad7af9a12cce3c8e416fb67bd6ec9207f0082d 192.168.0.130:6379@16379 slave
1a32d809c0b743bd83b5e1c277d5d201d0140b75 0 1568084031
000 2 connected
7ca218299c254b5da939f8e60a940ac8171adc27 192.168.0.22:6379@16379 master - 0 1568084030000
1 connected 0-5460
f0141816260ca5029c56333095f015c7a058f113 192.168.0.170:6379@16379 master - 0
1568084031992 3 connected 10923-16383
19b1a400815396c6223963b013ec934a657bdc52 192.168.0.161:6379@16379 slave
7ca218299c254b5da939f8e60a940ac8171adc27 0 1568084031
000 1 connected
```

   Write operations can only be performed on master nodes. The CRC16 of the key modulo 16384 is taken to compute what is the hash slot of a given key.

   As shown in the following, the value of **CRC16 (KEY) mode 16384** determines the hash slot that a given key is located at and redirects the client to the node where the hash slot is located at.

```
192.168.0.170:6379> set hello world
-> Redirected to slot [866] located at 192.168.0.22:6379
```

```
OK
192.168.0.22:6379> set happy day
OK
192.168.0.22:6379> set abc 123
-> Redirected to slot [7638] located at 192.168.0.85:6379
OK
192.168.0.85:6379> get hello
-> Redirected to slot [866] located at 192.168.0.22:6379
"world"
192.168.0.22:6379> get abc
-> Redirected to slot [7638] located at 192.168.0.85:6379
"123"
192.168.0.85:6379>
```

----End

Procedure (Windows)

Download the compilation package of the Redis client for Windows. (This is not the source code package.) Decompress the package in any directory, open the CLI tool **cmd.exe**, and go to the directory. Then, run the following command to access the DCS Redis instance:

redis-cli.exe -h *XXX* -p *6379*

XXX indicates the IP address of the DCS instance and **6379** is an example port number used for accessing the DCS instance. For details about how to obtain the IP address and port number, see **Viewing Details of a DCS Instance**. Change the address and port as required.

4.2.3 Access in Different Languages

4.2.3.1 Java

4.2.3.1.1 Connecting to Redis on Jedis (Java)

This section describes how to access a Redis instance on Jedis. For more information about how to use other Redis clients, visit **the Redis official website**.

Spring Data Redis is already integrated with **Jedis** and **Lettuce** for Spring Boot projects. Spring Boot 1.x is integrated with Jedis, and Spring Boot 2.x is integrated with Lettuce. To use Jedis in Spring Boot 2.x and later, you need to solve Lettuce dependency conflicts.

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- View the IP address and port number of the DCS Redis instance to be accessed.

  For details, see **Viewing Details of a DCS Instance**.

Pom Configuration

```
<!-- import spring-data-redis -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
```

```
<!--In Spring Boot 2.0, Lettuce is used by default. To use Jedis, solve dependency conflicts.-->
    <exclusions>
        <exclusion>
            <groupId>io.lettuce</groupId>
            <artifactId>lettuce-core</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<!--Jedis dependency>
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>3.10.0</version>
</dependency>
```

application.properties Configuration

- Single-node, master/standby, and Proxy Cluster

```
#Redis host
spring.redis.host=<host>
#Redis port
spring.redis.port=<port>
#Redis database number
spring.redis.database=0
#Redis password
spring.redis.password=<password>
#Redis read/write timeout
spring.redis.timeout=2000
#Whether to enable connection pooling
spring.redis.jedis.pool.enabled=true
#Minimum connections in the pool
spring.redis.jedis.pool.min-idle=50
#Maximum idle connections in the pool
spring.redis.jedis.pool.max-idle=200
#Maximum connections in the pool
spring.redis.jedis.pool.max-active=200
#Maximum amount of time a connection allocation should block before throwing an exception when the pool is exhausted. The default value -1 indicates to wait indefinitely.
spring.redis.jedis.pool.max-wait=3000
#Interval for checking and evicting idle connection. Default: 60s.
spring.redis.jedis.pool.time-between-eviction-runs=60S
```

- Redis Cluster

```
#Redis Cluster node connection information
spring.redis.cluster.nodes=<ip:port>,<ip:port>,<ip:port>
#Redis Cluster password
spring.redis.password=<password>
#Redis Cluster max. redirecting times
spring.redis.cluster.max-redirects=3
#Redis read/write timeout
spring.redis.timeout=2000
#Whether to enable connection pooling
spring.redis.jedis.pool.enabled=true
#Minimum connections in the pool
spring.redis.jedis.pool.min-idle=50
#Maximum idle connections in the pool
spring.redis.jedis.pool.max-idle=200
#Maximum connections in the pool
spring.redis.jedis.pool.max-active=200
#Maximum amount of time a connection allocation should block before throwing an exception when the pool is exhausted. The default value -1 indicates to wait indefinitely.
spring.redis.jedis.pool.max-wait=3000
#Interval for checking and evicting idle connections. Default: 60s.
spring.redis.jedis.pool.time-between-eviction-runs=60S
```

Bean Configuration

- Single-node, master/standby, and Proxy Cluster

```
import java.time.Duration;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisStandaloneConfiguration;
import org.springframework.data.redis.connection.jedis.JedisClientConfiguration;
import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;

import redis.clients.jedis.JedisPoolConfig;

@Configuration
public class RedisConfiguration {

    @Value("${redis.host}")
    private String redisHost;

    @Value("${redis.port:6379}")
    private Integer redisPort = 6379;

    @Value("${redis.database:0}")
    private Integer redisDatabase = 0;

    @Value("${redis.password:}")
    private String redisPassword;

    @Value("${redis.connect.timeout:3000}")
    private Integer redisConnectTimeout = 3000;

    @Value("${redis.read.timeout:2000}")
    private Integer redisReadTimeout = 2000;

    @Value("${redis.pool.minSize:50}")
    private Integer redisPoolMinSize = 50;

    @Value("${redis.pool.maxSize:200}")
    private Integer redisPoolMaxSize = 200;

    @Value("${redis.pool.maxWaitMillis:3000}")
    private Integer redisPoolMaxWaitMillis = 3000;

    @Value("${redis.pool.softMinEvictableIdleTimeMillis:1800000}")
    private Integer redisPoolSoftMinEvictableIdleTimeMillis = 30 * 60 * 1000;

    @Value("${redis.pool.timeBetweenEvictionRunsMillis:60000}")
    private Integer redisPoolBetweenEvictionRunsMillis = 60 * 1000;

    @Bean
    public RedisConnectionFactory redisConnectionFactory(JedisClientConfiguration
clientConfiguration) {

        RedisStandaloneConfiguration standaloneConfiguration = new RedisStandaloneConfiguration();
        standaloneConfiguration.setHostName(redisHost);
        standaloneConfiguration.setPort(redisPort);
        standaloneConfiguration.setDatabase(redisDatabase);
        standaloneConfiguration.setPassword(redisPassword);

        return new JedisConnectionFactory(standaloneConfiguration, clientConfiguration);
    }

    @Bean
    public JedisClientConfiguration clientConfiguration() {

        JedisClientConfiguration clientConfiguration = JedisClientConfiguration.builder()
                .connectTimeout(Duration.ofMillis(redisConnectTimeout))
                .readTimeout(Duration.ofMillis(redisReadTimeout))
                .usePooling().poolConfig(redisPoolConfig())
                .build();
        return clientConfiguration;
    }

    private JedisPoolConfig redisPoolConfig() {

        JedisPoolConfig poolConfig = new JedisPoolConfig();
        //Minimum connections in the pool
        poolConfig.setMinIdle(redisPoolMinSize);
        //Maximum idle connections in the pool
        poolConfig.setMaxIdle(redisPoolMaxSize);
        //Maximum total connections in the pool
        poolConfig.setMaxTotal(redisPoolMaxSize);
        //Wait when pool is exhausted? Set to true to wait. To validate setMaxWait, it has to be true.
        poolConfig.setBlockWhenExhausted(true);
        //Longest time to wait for connection after pool is exhausted. The default value -1 indicates to wait indefinitely.
        poolConfig.setMaxWaitMillis(redisPoolMaxWaitMillis);
        //Set to true to enable connectivity test on creating connections. Default: false.
        poolConfig.setTestOnCreate(false);
        //Set to true to enable connectivity test on borrowing connections. Default: false. Set to false for heavy-traffic services to reduce overhead.
        poolConfig.setTestOnBorrow(true);
        //Set to true to enable connectivity test on returning connections. Default: false. Set to false for heavy-traffic services to reduce overhead.
        poolConfig.setTestOnReturn(false);
        //Indicates whether to check for idle connections. If this is set to false, idle connections are not evicted.
        poolConfig.setTestWhileIdle(true);
        //Duration after which idle connections are evicted. If the idle duration is greater than this value and the maximum number of idle connections is reached, idle connections are directly evicted.
        poolConfig.setSoftMinEvictableIdleTimeMillis(redisPoolSoftMinEvictableIdleTimeMillis);
        //Disable MinEvictableIdleTimeMillis().
        poolConfig.setMinEvictableIdleTimeMillis(-1);
        //Interval for checking and evicting idle connections. Default: 60s.
        poolConfig.setTimeBetweenEvictionRunsMillis(redisPoolBetweenEvictionRunsMillis);
        return poolConfig;
    }
}
```

- Redis Cluster

```
import java.time.Duration;
import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisClusterConfiguration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisNode;
import org.springframework.data.redis.connection.jedis.JedisClientConfiguration;
import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;

import redis.clients.jedis.JedisPoolConfig;

@Configuration
public class RedisConfiguration {

    @Value("${redis.cluster.nodes}")
    private String redisClusterNodes;

    @Value("${redis.password:}")
    private String redisPassword;

    @Value("${redis.connect.timeout:3000}")
    private Integer redisConnectTimeout = 3000;

    @Value("${redis.read.timeout:2000}")
    private Integer redisReadTimeout = 2000;
    @Value("${redis.pool.minSize:50}")
    private Integer redisPoolMinSize = 50;

    @Value("${redis.pool.maxSize:200}")
    private Integer redisPoolMaxSize = 200;

    @Value("${redis.pool.maxWaitMillis:3000}")
    private Integer redisPoolMaxWaitMillis = 3000;

    @Value("${redis.pool.softMinEvictableIdleTimeMillis:1800000}")
    private Integer redisPoolSoftMinEvictableIdleTimeMillis = 30 * 60 * 1000;

    @Value("${redis.pool.timeBetweenEvictionRunsMillis:60000}")
    private Integer redisPoolBetweenEvictionRunsMillis = 60 * 1000;

    @Bean
    public RedisConnectionFactory redisConnectionFactory(JedisClientConfiguration
clientConfiguration) {

        RedisClusterConfiguration clusterConfiguration = new RedisClusterConfiguration();

        List<RedisNode> clusterNodes = new ArrayList<>();
        for (String clusterNodeStr : redisClusterNodes.split(",")) {
            String[] nodeInfo = clusterNodeStr.split(":");
            clusterNodes.add(new RedisNode(nodeInfo[0], Integer.valueOf(nodeInfo[1])));
        }
        clusterConfiguration.setClusterNodes(clusterNodes);

        clusterConfiguration.setPassword(redisPassword);
        clusterConfiguration.setMaxRedirects(3);

        return new JedisConnectionFactory(clusterConfiguration, clientConfiguration);
    }

    @Bean
    public JedisClientConfiguration clientConfiguration() {

        JedisClientConfiguration clientConfiguration = JedisClientConfiguration.builder()
                .connectTimeout(Duration.ofMillis(redisConnectTimeout))
                .readTimeout(Duration.ofMillis(redisReadTimeout))
                .usePooling().poolConfig(redisPoolConfig())
                .build();

        return clientConfiguration;
    }

    private JedisPoolConfig redisPoolConfig() {

        JedisPoolConfig poolConfig = new JedisPoolConfig();
        //Minimum connections in the pool
        poolConfig.setMinIdle(redisPoolMinSize);
        //Maximum idle connections in the pool
        poolConfig.setMaxIdle(redisPoolMaxSize);
        //Maximum total connections in the pool
        poolConfig.setMaxTotal(redisPoolMaxSize);
        //Wait when pool is exhausted? Set to true to wait. To validate setMaxWait, it has to be true.
        poolConfig.setBlockWhenExhausted(true);
        //Longest time to wait for connection after pool is exhausted. The default value -1 indicates to
wait indefinitely.
        poolConfig.setMaxWaitMillis(redisPoolMaxWaitMillis);
        //Set to true to enable connectivity test on creating connections. Default: false.
        poolConfig.setTestOnCreate(false);
        //Set to true to enable connectivity test on borrowing connections. Default: false. Set to false for
heavy-traffic services to reduce overhead.
        poolConfig.setTestOnBorrow(true);
        //Set to true to enable connectivity test on returning connections. Default: false. Set to false for
heavy-traffic services to reduce overhead.
        poolConfig.setTestOnReturn(false);
        //Indicates whether to check for idle connections. If this is set to false, idle connections are not
evicted.
        poolConfig.setTestWhileIdle(true);
        //Duration after which idle connections are evicted. If the idle duration is greater than this value
and the maximum number of idle connections is reached, idle connections are directly evicted.
        poolConfig.setSoftMinEvictableIdleTimeMillis(redisPoolSoftMinEvictableIdleTimeMillis);
        //Disable MinEvictableIdleTimeMillis().
        poolConfig.setMinEvictableIdleTimeMillis(-1);
        //Interval for checking and evicting idle connections. Default: 60s.
        poolConfig.setTimeBetweenEvictionRunsMillis(redisPoolBetweenEvictionRunsMillis);
        return poolConfig;
    }
}
```

(Optional) Configuring SSL Connections

If SSL is enabled for the instance, use the following content to replace the JedisClientConfiguration construction method clientConfiguration() in **Bean Configuration** for connecting to the instance with SSL. For details about whether your DCS Redis instances support SSL, see **Transmitting DCS Redis Data with Encryption Using SSL**.

```
@Bean
public JedisClientConfiguration clientConfiguration() throws Exception {
    JedisClientConfiguration.JedisClientConfigurationBuilder configurationBuilder
        = JedisClientConfiguration.builder()
        .connectTimeout(Duration.ofMillis(redisConnectTimeout))
        .readTimeout(Duration.ofMillis(redisReadTimeout));

    configurationBuilder.usePooling().poolConfig(redisPoolConfig());
    configurationBuilder.useSsl().sslSocketFactory(getTrustStoreSslSocketFactory());
    return configurationBuilder.build();
}

private SSLSocketFactory getTrustStoreSslSocketFactory() throws Exception{
    //Load the CA certificate in the user-defined path based on service requirements.
    CertificateFactory cf = CertificateFactory.getInstance("X.509");
    Certificate ca;
    try (InputStream is = new FileInputStream("./ca.crt")) {
        ca = cf.generateCertificate(is);
    }

    //Create keystore.
    String keyStoreType = KeyStore.getDefaultType();
    KeyStore keyStore = KeyStore.getInstance(keyStoreType);
    keyStore.load(null, null);
    keyStore.setCertificateEntry("ca", ca);

    //Create TrustManager.
    TrustManagerFactory trustManagerFactory = TrustManagerFactory.getInstance(
        TrustManagerFactory.getDefaultAlgorithm());
    trustManagerFactory.init(keyStore);

    //Create SSLContext.
    SSLContext context = SSLContext.getInstance("TLS");
    context.init(null, trustManagerFactory.getTrustManagers(), new SecureRandom());
    return context.getSocketFactory();
}
```

Parameter Description

Table 4-2 RedisStandaloneConfiguration parameters

Parameter	Default Value	Description
hostName	localhost	IP address for connecting to a DCS Redis instance
port	6379	Port number
database	0	Database number. Default: 0.
password	-	Password

Table 4-3 RedisClusterConfiguration parameters

Parameter	Description
clusterNodes	Cluster node connection information, including the node IP address and port number
maxRedirects	Maximum redirecting times
password	Password

Table 4-4 JedisPoolConfig parameters

Parameter	Default Value	Description
minIdle	-	Minimum connections in the connection pool
maxIdle	-	Maximum idle connections in the connection pool
maxTotal	-	Maximum total connections in the connection pool
blockWhenExhausted	true	Indicates whether to wait after the connection pool is exhausted. **true**: Wait. **false**: Do not wait. To validate **maxWaitMillis**, this parameter must be set to **true**.
maxWaitMillis	-1	Maximum amount of time (in milliseconds) to wait for connection after the connection pool is exhausted. The default value **-1** indicates to wait indefinitely.

Parameter	Default Value	Description
testOnCreate	false	Indicates whether to enable connectivity test on creating connections. **false**: Disable. **true**: Enable.
testOnBorrow	false	Indicates whether to enable connectivity test on obtaining connections. **false**: Disable. **true**: Enable. For heavy-traffic services, set this parameter to **false** to reduce overhead.
testOnReturn	false	Indicates whether to enable connectivity test on returning connections. **false**: Disable. **true**: Enable. For heavy-traffic services, set this parameter to **false** to reduce overhead.
testWhileIdle	false	Indicates whether to check for idle connections. If this parameter is set to **false**, idle connections are not evicted. Recommended value: **true**.
softMinEvictableIdleTimeMillis	1800000	Duration (in milliseconds) after which idle connections are evicted. If the idle duration is greater than this value and the maximum number of idle connections is reached, idle connections are directly evicted.
minEvictableIdleTimeMillis	60000	Minimum amount of time (in milliseconds) a connection may remain idle in the pool before it is eligible for eviction. The recommended value is **-1**, indicating that **softMinEvictableIdleTimeMillis** is used instead.
timeBetweenEvictionRunsMillis	60000	Interval (in milliseconds) for checking and evicting idle connections.

Table 4-5 JedisClientConfiguration parameters

Parameter	Default Value	Description
connectTimeout	2000	Connection timeout interval, in milliseconds.
readTimeout	2000	Timeout interval for waiting for a response, in milliseconds.
poolConfig	-	Pool configurations. For details, see **JedisPoolConfig**.

Suggestion for Configuring DCS Instances

- Connection pool configuration

  NOTE

  The following calculation is applicable only to common service scenarios. You can customize it based on your service requirements.

  There is no standard connection pool size. You can configure one based on your service traffic. The following formulas are for reference:

  - Minimum number of connections = (QPS of a single node accessing Redis)/(1000 ms/Average time spent on a single command)
  - Maximum number of connections = (QPS of a single node accessing Redis)/(1000 ms/Average time spent on a single command) x 150%

  For example, if the QPS of a service application is about 10,000, each request needs to access Redis 10 times (that is, 100,000 accesses to Redis every second), and the service application has 10 hosts, the calculation is as follows:

  QPS of a single node accessing Redis = 100,000/10 = 10,000

  Average time spent on a single command = 20 ms (Redis takes 5 ms to 10 ms to process a single command under normal conditions. If network jitter occurs, it takes 15 ms to 20 ms.)

  Minimum number of connections = 10,000/(1000 ms/20 ms) = 200

  Maximum number of connections = 10,000/(1000 ms/20 ms) x 150% = 300

4.2.3.1.2 Connecting to Redis on Lettuce (Java)

This section describes how to access a Redis instance on Lettuce. For more information about how to use other Redis clients, visit **the Redis official website**.

Spring Data Redis is already integrated with **Jedis** and **Lettuce** for Spring Boot projects. In addition, Spring Boot 1.x is integrated with Jedis, and Spring Boot 2.x with Lettuce. Therefore, you do not need to import Lettuce in Spring Boot 2.x and later projects.

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- View the IP address and port number of the DCS Redis instance to be accessed.

  For details, see **Viewing Details of a DCS Instance**.

Pom Configuration

```
<!-- Enable Spring Data Redis, Lettuce-supported SDK is integrated by default -->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

application.properties Configuration

- Single-node, master/standby, and Proxy Cluster

```
#Redis host
spring.redis.host=<host>
#Redis port
```

```
spring.redis.port=<port>
#Redis database number
spring.redis.database=0
#Redis password
spring.redis.password=<password>
#Redis read/write timeout
spring.redis.timeout=2000
```

- Redis Cluster

```
#Redis Cluster node information
spring.redis.cluster.nodes=<ip:port>,<ip:port>,<ip:port>
#Redis Cluster max redirecting times
spring.redis.cluster.max-redirects=3
#Redis Cluster node password
spring.redis.password=<password>
#Redis Cluster timeout
spring.redis.timeout=2000
#Enable adaptive topology refresh
spring.redis.lettuce.cluster.refresh.adaptive=true
#Enable topology refresh every 10 seconds
spring.redis.lettuce.cluster.refresh.period=10S
```

Bean Configuration

- Single-node, master/standby, and Proxy Cluster

```
import java.time.Duration;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisStandaloneConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;

import io.lettuce.core.ClientOptions;
import io.lettuce.core.SocketOptions;

/**
* Lettuce non-pooling configuration (use either this or the application.properties configuration)
*/
@Configuration
public class RedisConfiguration {

    @Value("${redis.host}")
    private String redisHost;

    @Value("${redis.port:6379}")
    private Integer redisPort = 6379;

    @Value("${redis.database:0}")
    private Integer redisDatabase = 0;

    @Value("${redis.password:}")
    private String redisPassword;

    @Value("${redis.connect.timeout:2000}")
    private Integer redisConnectTimeout = 2000;

    @Value("${redis.read.timeout:2000}")
    private Integer redisReadTimeout = 2000;

    @Bean
    public RedisConnectionFactory redisConnectionFactory(LettuceClientConfiguration
clientConfiguration) {

        RedisStandaloneConfiguration standaloneConfiguration = new RedisStandaloneConfiguration();
        standaloneConfiguration.setHostName(redisHost);
        standaloneConfiguration.setPort(redisPort);
        standaloneConfiguration.setDatabase(redisDatabase);
        standaloneConfiguration.setPassword(redisPassword);

        LettuceConnectionFactory connectionFactory = new
LettuceConnectionFactory(standaloneConfiguration, clientConfiguration);
        connectionFactory.setDatabase(redisDatabase);
        return connectionFactory;
    }

    @Bean
    public LettuceClientConfiguration clientConfiguration() {

        SocketOptions socketOptions =
SocketOptions.builder().connectTimeout(Duration.ofMillis(redisConnectTimeout)).build();

        ClientOptions clientOptions = ClientOptions.builder()
                .autoReconnect(true)
                .pingBeforeActivateConnection(true)
                .cancelCommandsOnReconnectFailure(false)
                .disconnectedBehavior(ClientOptions.DisconnectedBehavior.ACCEPT_COMMANDS)
                .socketOptions(socketOptions)
                .build();

        LettuceClientConfiguration clientConfiguration = LettuceClientConfiguration.builder()
                .commandTimeout(Duration.ofMillis(redisReadTimeout))
                .clientOptions(clientOptions)
                .build();
        return clientConfiguration;
    }
}
```

- Pooling configuration for single-node, master/standby, and Proxy Cluster instances

  Enable the pooling component

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-pool2</artifactId>
    <version>2.11.1</version>
</dependency>
```

  Code

```
import java.time.Duration;

import org.apache.commons.pool2.impl.GenericObjectPoolConfig;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisStandaloneConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;
import org.springframework.data.redis.connection.lettuce.LettucePoolingClientConfiguration;

import io.lettuce.core.ClientOptions;
import io.lettuce.core.SocketOptions;

/**
* Lettuce pooling configuration
*/
@Configuration
public class RedisPoolConfiguration {
    @Value("${redis.host}")
    private String redisHost;

    @Value("${redis.port:6379}")
    private Integer redisPort = 6379;

    @Value("${redis.database:0}")
    private Integer redisDatabase = 0;
    @Value("${redis.password:}")
    private String redisPassword;

    @Value("${redis.connect.timeout:2000}")
    private Integer redisConnectTimeout = 2000;

    @Value("${redis.read.timeout:2000}")
    private Integer redisReadTimeout = 2000;

    @Value("${redis.pool.minSize:50}")
    private Integer redisPoolMinSize = 50;

    @Value("${redis.pool.maxSize:200}")
    private Integer redisPoolMaxSize = 200;

    @Value("${redis.pool.maxWaitMillis:2000}")
    private Integer redisPoolMaxWaitMillis = 2000;

    @Value("${redis.pool.softMinEvictableIdleTimeMillis:1800000}")
    private Integer redisPoolSoftMinEvictableIdleTimeMillis = 30 * 60 * 1000;

    @Value("${redis.pool.timeBetweenEvictionRunsMillis:60000}")
    private Integer redisPoolBetweenEvictionRunsMillis = 60 * 1000;

    @Bean
    public RedisConnectionFactory redisConnectionFactory(LettuceClientConfiguration
clientConfiguration) {

        RedisStandaloneConfiguration standaloneConfiguration = new RedisStandaloneConfiguration();
        standaloneConfiguration.setHostName(redisHost);
        standaloneConfiguration.setPort(redisPort);
        standaloneConfiguration.setDatabase(redisDatabase);
        standaloneConfiguration.setPassword(redisPassword);

        LettuceConnectionFactory connectionFactory = new
LettuceConnectionFactory(standaloneConfiguration, clientConfiguration);
        connectionFactory.setDatabase(redisDatabase);
        //Disable sharing native connection before enabling pooling
        connectionFactory.setShareNativeConnection(false);
        return connectionFactory;
    }

    @Bean
    public LettuceClientConfiguration clientConfiguration() {

        SocketOptions socketOptions =
SocketOptions.builder().connectTimeout(Duration.ofMillis(redisConnectTimeout)).build();

        ClientOptions clientOptions = ClientOptions.builder()
                .autoReconnect(true)
                .pingBeforeActivateConnection(true)
                .cancelCommandsOnReconnectFailure(false)
                .disconnectedBehavior(ClientOptions.DisconnectedBehavior.ACCEPT_COMMANDS)
                .socketOptions(socketOptions)
                .build();

        LettucePoolingClientConfiguration poolingClientConfiguration =
LettucePoolingClientConfiguration.builder()
                .poolConfig(redisPoolConfig())
                .commandTimeout(Duration.ofMillis(redisReadTimeout))
                .clientOptions(clientOptions)
                .build();
        return poolingClientConfiguration;
    }

    private GenericObjectPoolConfig redisPoolConfig() {
        GenericObjectPoolConfig poolConfig = new GenericObjectPoolConfig();
        //Minimum idle connections in the pool
        poolConfig.setMinIdle(redisPoolMinSize);
        //Maximum idle connections in the pool
        poolConfig.setMaxIdle(redisPoolMaxSize);
        //Maximum total connections in the pool
        poolConfig.setMaxTotal(redisPoolMaxSize);
        //Wait when pool is exhausted? Set to true to wait. To validate setMaxWait, it has to be true.
        poolConfig.setBlockWhenExhausted(true);
        //Max allowed time to wait for connection after pool is exhausted. The default value -1 indicates
to wait indefinitely.
        poolConfig.setMaxWait(Duration.ofMillis(redisPoolMaxWaitMillis));
        //Set to true to enable connectivity test on creating connections. Default: false.
        poolConfig.setTestOnCreate(false);
        //Set to true to enable connectivity test on borrowing connections. Default: false. Set to false for
heavy-traffic services to reduce overhead.
        poolConfig.setTestOnBorrow(true);
        //Set to true to enable connectivity test on returning connections. Default: false. Set to false for
heavy-traffic services to reduce overhead.
        poolConfig.setTestOnReturn(false);
        //Indicates whether to check for idle connections. If this is set to false, idle connections are not
evicted.
        poolConfig.setTestWhileIdle(true);
        //Idle duration after which a connection is evicted. If the actual duration is greater than this
value and the maximum number of idle connections is reached, idle connections are directly evicted.

poolConfig.setSoftMinEvictableIdleTime(Duration.ofMillis(redisPoolSoftMinEvictableIdleTimeMillis));
        //Disable eviction policy MinEvictableIdleTimeMillis().
        poolConfig.setMinEvictableIdleTime(Duration.ofMillis(-1));
        //Interval for checking and evicting idle connections. Default: 60s.
        poolConfig.setTimeBetweenEvictionRuns(Duration.ofMillis(redisPoolBetweenEvictionRunsMillis));
        return poolConfig;
    }
}
```

- Configuration for Redis Cluster instances

```
import java.time.Duration;
import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisClusterConfiguration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisNode;
import org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;

import io.lettuce.core.ClientOptions;
import io.lettuce.core.SocketOptions;
import io.lettuce.core.cluster.ClusterClientOptions;
import io.lettuce.core.cluster.ClusterTopologyRefreshOptions;

/**
* Lettuce Cluster non-pooling configuration (use either this or the application.properties configuration)
*/
@Configuration
public class RedisConfiguration {

    @Value("${redis.cluster.nodes}")
    private String redisClusterNodes;

    @Value("${redis.cluster.maxDirects:3}")
    private Integer redisClusterMaxDirects;

    @Value("${redis.password:}")
    private String redisPassword;

    @Value("${redis.connect.timeout:2000}")
    private Integer redisConnectTimeout = 2000;
    @Value("${redis.read.timeout:2000}")
    private Integer redisReadTimeout = 2000;

    @Value("${redis.cluster.topology.refresh.period.millis:10000}")
    private Integer redisClusterTopologyRefreshPeriodMillis = 10000;

    @Bean
    public RedisConnectionFactory redisConnectionFactory(LettuceClientConfiguration
clientConfiguration) {

        RedisClusterConfiguration clusterConfiguration = new RedisClusterConfiguration();

        List<RedisNode> clusterNodes = new ArrayList<>();
        for (String clusterNodeStr : redisClusterNodes.split(",")) {
            String[] nodeInfo = clusterNodeStr.split(":");
            clusterNodes.add(new RedisNode(nodeInfo[0], Integer.valueOf(nodeInfo[1])));
        }
        clusterConfiguration.setClusterNodes(clusterNodes);

        clusterConfiguration.setPassword(redisPassword);
        clusterConfiguration.setMaxRedirects(redisClusterMaxDirects);

        LettuceConnectionFactory connectionFactory = new
LettuceConnectionFactory(clusterConfiguration, clientConfiguration);
        return connectionFactory;
    }

    @Bean
    public LettuceClientConfiguration clientConfiguration() {

        SocketOptions socketOptions =
SocketOptions.builder().connectTimeout(Duration.ofMillis(redisConnectTimeout)).build();

        ClusterTopologyRefreshOptions topologyRefreshOptions =
ClusterTopologyRefreshOptions.builder()
                .enableAllAdaptiveRefreshTriggers()
                .enablePeriodicRefresh(Duration.ofMillis(redisClusterTopologyRefreshPeriodMillis))
                .build();

        ClusterClientOptions clientOptions = ClusterClientOptions.builder()
                .autoReconnect(true)
                .pingBeforeActivateConnection(true)
                .cancelCommandsOnReconnectFailure(false)
                .disconnectedBehavior(ClientOptions.DisconnectedBehavior.ACCEPT_COMMANDS)
                .socketOptions(socketOptions)
                .topologyRefreshOptions(topologyRefreshOptions)
                .build();

        LettuceClientConfiguration clientConfiguration = LettuceClientConfiguration.builder()
                .commandTimeout(Duration.ofMillis(redisReadTimeout))
                .clientOptions(clientOptions)
                .build();
        return clientConfiguration;
    }
}
```

- Pooling configuration for Redis Cluster instances

  Enable the pooling component

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-pool2</artifactId>
    <version>2.11.1</version>
</dependency>
```

  Code

```
import java.time.Duration;
import java.util.ArrayList;
import java.util.List;
import org.apache.commons.pool2.impl.GenericObjectPoolConfig;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisClusterConfiguration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisNode;
import org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;
import org.springframework.data.redis.connection.lettuce.LettucePoolingClientConfiguration;

import io.lettuce.core.ClientOptions;
import io.lettuce.core.SocketOptions;
import io.lettuce.core.cluster.ClusterClientOptions;
import io.lettuce.core.cluster.ClusterTopologyRefreshOptions;

/**
* Lettuce pooling configuration
*/
@Configuration
public class RedisPoolConfiguration {

    @Value("${redis.cluster.nodes}")
    private String redisClusterNodes;

    @Value("${redis.cluster.maxDirects:3}")
    private Integer redisClusterMaxDirects;

    @Value("${redis.password:}")
    private String redisPassword;

    @Value("${redis.connect.timeout:2000}")
    private Integer redisConnectTimeout = 2000;

    @Value("${redis.read.timeout:2000}")
    private Integer redisReadTimeout = 2000;

    @Value("${redis.cluster.topology.refresh.period.millis:10000}")
    private Integer redisClusterTopologyRefreshPeriodMillis = 10000;

    @Value("${redis.pool.minSize:50}")
    private Integer redisPoolMinSize = 50;

    @Value("${redis.pool.maxSize:200}")
    private Integer redisPoolMaxSize = 200;

    @Value("${redis.pool.maxWaitMillis:2000}")
    private Integer redisPoolMaxWaitMillis = 2000;

    @Value("${redis.pool.softMinEvictableIdleTimeMillis:1800000}")
    private Integer redisPoolSoftMinEvictableIdleTimeMillis = 30 * 60 * 1000;

    @Value("${redis.pool.timeBetweenEvictionRunsMillis:60000}")
    private Integer redisPoolBetweenEvictionRunsMillis = 60 * 1000;

    @Bean
    public RedisConnectionFactory redisConnectionFactory(LettuceClientConfiguration
clientConfiguration) {

        RedisClusterConfiguration clusterConfiguration = new RedisClusterConfiguration();

        List<RedisNode> clusterNodes = new ArrayList<>();
        for (String clusterNodeStr : redisClusterNodes.split(",")) {
            String[] nodeInfo = clusterNodeStr.split(":");
            clusterNodes.add(new RedisNode(nodeInfo[0], Integer.valueOf(nodeInfo[1])));
        }
        clusterConfiguration.setClusterNodes(clusterNodes);

        clusterConfiguration.setPassword(redisPassword);
        clusterConfiguration.setMaxRedirects(redisClusterMaxDirects);

        LettuceConnectionFactory connectionFactory = new
LettuceConnectionFactory(clusterConfiguration, clientConfiguration);
        //Disable native connection sharing before validating connection pool
        connectionFactory.setShareNativeConnection(false);
        return connectionFactory;
    }

    @Bean
    public LettuceClientConfiguration clientConfiguration() {

        SocketOptions socketOptions =
SocketOptions.builder().connectTimeout(Duration.ofMillis(redisConnectTimeout)).build();

        ClusterTopologyRefreshOptions topologyRefreshOptions =
ClusterTopologyRefreshOptions.builder()
                .enableAllAdaptiveRefreshTriggers()
                .enablePeriodicRefresh(Duration.ofMillis(redisClusterTopologyRefreshPeriodMillis))
                .build();

        ClusterClientOptions clientOptions = ClusterClientOptions.builder()
                .autoReconnect(true)
                .pingBeforeActivateConnection(true)
                .cancelCommandsOnReconnectFailure(false)
                .disconnectedBehavior(ClientOptions.DisconnectedBehavior.ACCEPT_COMMANDS)
                .socketOptions(socketOptions)
                .topologyRefreshOptions(topologyRefreshOptions)
                .build();

        LettucePoolingClientConfiguration clientConfiguration =
LettucePoolingClientConfiguration.builder()
                .poolConfig(poolConfig())
                .commandTimeout(Duration.ofMillis(redisReadTimeout))
                .clientOptions(clientOptions)
                .build();
        return clientConfiguration;
    }

    private GenericObjectPoolConfig poolConfig() {
        GenericObjectPoolConfig poolConfig = new GenericObjectPoolConfig();
        //Minimum connections in the pool
        poolConfig.setMinIdle(redisPoolMinSize);
        //Maximum idle connections in the pool
        poolConfig.setMaxIdle(redisPoolMaxSize);
        //Maximum total connections in the pool
        poolConfig.setMaxTotal(redisPoolMaxSize);
        //Wait when pool is exhausted? Set to true to wait. To validate setMaxWait, it has to be true.
        poolConfig.setBlockWhenExhausted(true);
        //Max allowed time to wait for connection after pool is exhausted. The default value -1 indicates
to wait indefinitely.
        poolConfig.setMaxWait(Duration.ofMillis(redisPoolMaxWaitMillis));
        //Set to true to enable connectivity test on creating connections. Default: false.
        poolConfig.setTestOnCreate(false);
        //Set to true to enable connectivity test on borrowing connections. Default: false. Set to false for
heavy-traffic services to reduce overhead.
        poolConfig.setTestOnBorrow(true);
        //Set to true to enable connectivity test on returning connections. Default: false. Set to false for
heavy-traffic services to reduce overhead.
        poolConfig.setTestOnReturn(false);
        //Indicates whether to check for idle connections. If this is set to false, idle connections are not
evicted.
        poolConfig.setTestWhileIdle(true);
        //Disable connection closure when the minimum idle time is reached.
        poolConfig.setMinEvictableIdleTime(Duration.ofMillis(-1));
        //Idle duration before a connection being evicted. If the actual duration is greater than this value
and the maximum number of idle connections is reached, idle connections are directly evicted.
MinEvictableIdleTimeMillis (default eviction policy) is no longer used.
```

```
poolConfig.setSoftMinEvictableIdleTime(Duration.ofMillis(redisPoolSoftMinEvictableIdleTimeMillis));
        //Interval for checking and evicting idle connections. Default: 60s.
        poolConfig.setTimeBetweenEvictionRuns(Duration.ofMillis(redisPoolBetweenEvictionRunsMillis));

        return poolConfig;
    }

}
```

(Optional) Configuring SSL Connections

If SSL is enabled for an instance, to access it using SSL connections, use the following content to replace the **LettuceClientConfiguration** construction method **clientConfiguration()** in **Bean Configuration**. For details about whether your DCS Redis instances support SSL, see **Transmitting DCS Redis Data with Encryption Using SSL**.

- Single-node, master/standby, and Proxy Cluster

```
@Bean
public LettuceClientConfiguration clientConfiguration() {

    SocketOptions socketOptions = 
SocketOptions.builder().connectTimeout(Duration.ofMillis(redisConnectTimeout)).build();

    SslOptions sslOptions = SslOptions.builder()
        .trustManager(new File(certificationPath))
        .build();

    ClientOptions clientOptions = ClientOptions.builder()
        .sslOptions(sslOptions)
        .autoReconnect(true)
        .pingBeforeActivateConnection(true)
        .cancelCommandsOnReconnectFailure(false)
        .disconnectedBehavior(ClientOptions.DisconnectedBehavior.ACCEPT_COMMANDS)
        .socketOptions(socketOptions)
        .build();

    LettuceClientConfiguration clientConfiguration = LettuceClientConfiguration.builder()
        .commandTimeout(Duration.ofMillis(redisReadTimeout))
        .clientOptions(clientOptions)
        .useSsl()
        .build();

    return clientConfiguration;
}
```

- Redis Cluster

```
@Bean
public LettuceClientConfiguration clientConfiguration() {

    SocketOptions socketOptions = 
SocketOptions.builder().connectTimeout(Duration.ofMillis(redisConnectTimeout)).build();

    SslOptions sslOptions = SslOptions.builder()
        .trustManager(new File(certificationPath))
        .build();

    ClusterTopologyRefreshOptions topologyRefreshOptions = ClusterTopologyRefreshOptions.builder()
        .enableAllAdaptiveRefreshTriggers()
        .enablePeriodicRefresh(Duration.ofMillis(redisClusterTopologyRefreshPeriodMillis))
        .build();

    ClusterClientOptions clientOptions = ClusterClientOptions.builder()
        .sslOptions(sslOptions)
        .autoReconnect(true)
        .pingBeforeActivateConnection(true)
        .cancelCommandsOnReconnectFailure(false)
        .disconnectedBehavior(ClientOptions.DisconnectedBehavior.ACCEPT_COMMANDS)
        .socketOptions(socketOptions)
        .topologyRefreshOptions(topologyRefreshOptions)
        .build();

    LettuceClientConfiguration clientConfiguration = LettuceClientConfiguration.builder()
        .commandTimeout(Duration.ofMillis(redisReadTimeout))
        .clientOptions(clientOptions)
        .useSsl()
        .build();

    return clientConfiguration;
}
```

Parameter Description

Table 4-6 LettuceConnectionFactory parameters

Parameter	Type	Default Value	Description
configuration	RedisConfiguration	-	Redis connection configuration. Two subsclasses: ● RedisStandaloneConfiguration ● RedisClusterConfiguration
clientConfiguration	LettuceClientConfiguration	-	Client configuration parameter. Common subclass: LettucePoolingClientConfiguration
shareNativeConnection	boolean	true	Indicates whether to share native connections. Set to **true** to share. Set to **false** to enable connection pooling.

Table 4-7 RedisStandaloneConfiguration parameters

Parameter	Default Value	Description
hostName	localhost	IP address for connecting to a DCS Redis instance
port	6379	Port number
database	0	Database subscript
password	-	Password

Table 4-8 RedisClusterConfiguration parameters

Parameter	Description
clusterNodes	Cluster node connection information, including the node IP address and port number

Parameter	Description
maxRedirects	Maximum redirecting times. Recommended value: **3**.
password	Password

Table 4-9 LettuceClientConfiguration parameters

Parameter	Type	Default Value	Description
timeout	Duration	60s	Command timeout: Recommended: **2s**.
clientOptions	ClientOptions	-	Configuration options.

Table 4-10 LettucePoolingClientConfiguration parameters

Parameter	Type	Default Value	Description
timeout	Duration	60s	Command timeout: Recommended: **2s**.
clientOptions	ClientOptions	-	Configuration options.
poolConfig	GenericObjectPoolConfig	-	Connection pool configuration.

Table 4-11 ClientOptions parameters

Parameter	Type	Default Value	Description
autoReconnect	boolean	true	Indicates whether to automatically reconnect after disconnection. Recommended: **true**.
pingBeforeActivateConnection	boolean	true	Indicates whether to test connectivity on established connections. Recommended: **true**.

Parameter	Type	Default Value	Description
cancelCommandsOnReconnectFailure	boolean	true	Indicates whether to cancel commands after a failed reconnection attempt. Recommended: **false**.
disconnectedBehavior	DisconnectedBehavior	DisconnectedBehavior.DEFAULT	Indicates what to do when a connection drops. Recommended: **ACCEPT_COMMANDS**. ● **DEFAULT**: When **autoReconnect** is set **true**, commands are allowed to wait in queue. When **autoReconnect** is set to **false**, commands are not allowed to wait in queue. ● **ACCEPT_COMMANDS**: Allow commands to wait in queue. ● **REJECT_COMMANDS**: Do not allow commands to wait in queue.
socketOptions	SocketOptions	-	Socket configuration.

Table 4-12 SocketOptions parameters

Parameter	Default Value	Description
connectTimeout	10s	Connection timeout. Recommended: **2s**.

Table 4-13 GenericObjectPoolConfig parameters

Parameter	Default Value	Description
minIdle	-	Minimum connections in the pool.
maxIdle	-	Maximum idle connections in the connection pool.
maxTotal	-	Maximum total connections in the connection pool.

Parameter	Default Value	Description
blockWhenExhausted	true	Indicates whether to wait after the connection pool is exhausted. **true**: Wait. **false**: Do not wait. To validate **maxWaitMillis**, this parameter must be set to **true**.
maxWaitMillis	-1	Maximum amount of time a connection allocation should block before throwing an exception when the pool is exhausted. The default value **–1** indicates to wait indefinitely.
testOnCreate	false	Set to true to enable connectivity test on creating connections. Default: **false**.
testOnBorrow	false	Set to true to enable connectivity test on borrowing connections. Default: **false**. Set to false for heavy-traffic services to reduce overhead.
testOnReturn	false	Set to **true** to enable connectivity test on returning connections. Default: **false**. Set to **false** for heavy-traffic services to reduce overhead.
testWhileIdle	false	Indicates whether to check for idle connections. If this parameter is set to **false**, idle connections are not evicted. Recommended value: **true**.
softMinEvictableIdleTimeMillis	1800000	Duration after which idle connections are evicted. If the idle duration is greater than this value and the maximum number of idle connections is reached, idle connections are directly evicted.
minEvictableIdleTimeMillis	60000	An eviction policy, set to **–1** (suggested) to disable it. Use **softminEvictableIdleTimeMillis** instead.
timeBetweenEvictionRunsMillis	60000	Eviction interval, in milliseconds.

Suggestion for Configuring DCS Instances

- Pooling connection

  Different from Jedis's BIO, the bottom layer of Lettuce communicates with Redis Server based on Netty's NIO. Combining persistent connections and queues, Lettuce sends and receives multiple requests and responses spontaneously with sequential sending and receiving features of TCP. A single connection supports 3000 to 5000 QPS, but you are not advised to allow more than 3000 QPS in production systems. Pooling is not supported by Lettuce, and is disabled by default in Spring Boot. To enable pooling, validate the commons-pool2 dependency and disable native connection sharing.

By default, each Lettuce connection needs two thread pools, I/O thread pool and computation thread pool, to support I/O event reading and asynchronous event processing. If you configure connection pooling, each connection creates two thread pools, consuming high memory resources. **Lettuce is strong at processing single connections based on its bottom-layer implementation, so you are not advised to use Lettuce with pooling.**

- Topology refresh

  When connecting to a Redis Cluster instance, Lettuce randomly sends **cluster nodes** to the node list during initialization to obtain the distribution of cluster slots. Cluster topology structure changes when the cluster capacity is increased or decreased or a master/standby switchover occurs. Lettuce does not detect such changes by default. You can enable detection with the following configurations:

  – **application.properties configuration**

  ```
  #Enable adaptive topology refresh.
  spring.redis.lettuce.cluster.refresh.adaptive=true
  #Enable topology refresh every 10 seconds.
  spring.redis.lettuce.cluster.refresh.period=10S
  ```

  – **API configuration**

  ```
  ClusterTopologyRefreshOptions topologyRefreshOptions =
  ClusterTopologyRefreshOptions.builder()
      .enableAllAdaptiveRefreshTriggers()
      .enablePeriodicRefresh(Duration.ofMillis(redisClusterTopologyRefreshPeriodMillis))
      .build();

  ClusterClientOptions clientOptions = ClusterClientOptions.builder()
          ...
          ...
          .topologyRefreshOptions(topologyRefreshOptions)
          .build();
  ```

- Blast radius

  The bottom layer of Lettuce uses a combination of single persistent connection and request queue. Once network jitter or intermittent disconnection occurs or connection times out, all requests are affected. Especially when connection times out, an attempt is made to resend TCP pockets until timeout and connection drops. Requests do not recover until connections are reestablished. Requests accumulate during resending attempts. If upper-layer services time out in batches, or the resending timeout is too long in some OSs' kernels, the service system remains unavailable for a long time. **Therefore, you are advised to use Jedis instead of Lettuce.**

4.2.3.1.3 Connecting to Redis on Redisson (Java)

This section describes how to access a Redis instance on Redisson. For more information about how to use other Redis clients, visit **the Redis official website**.

For Spring Boot projects, Spring Data Redis is already integrated with **Jedis** and **Lettuce**, but does not support Redisson. **Redisson** provides the redisson-spring-boot-starter component (https://mvnrepository.com/artifact/org.redisson/redisson) that can be used with Spring Boot.

Spring Boot 1.x is integrated with Jedis, and Spring Boot 2.x is integrated with Lettuce.

NOTE

- If a password was set during DCS Redis instance creation, configure the password for connecting to Redis using Redisson. Do not hard code the plaintext password.
- To connect to a single-node or Proxy Cluster instance, use the **useSingleServer** method of the **SingleServerConfig** object of Redisson. To connect to a master/standby instance, use the **useMasterSlaveServers** method of the **MasterSlaveServersConfig** object of Redisson. To connect to a Redis Cluster instance, use the **useClusterServers** method of the **ClusterServersConfig** object.

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- View the IP address and port number of the DCS Redis instance to be accessed.

  For details, see **Viewing Details of a DCS Instance**.

Pom Configuration

```
<!-- spring-data-redis -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
    <exclusions>
        <!--Lettuce is integrated in Spring Boot 2.x by default. This dependency needs to be deleted. -->
        <exclusion>
            <artifactId>lettuce-core</artifactId>
            <groupId>io.lettuce</groupId>
        </exclusion>
    </exclusions>
</dependency>
<!--Redisson's adaptation package for Spring Boot-->
<dependency>
    <groupId>org.redisson</groupId>
    <artifactId>redisson-spring-boot-starter</artifactId>
    <version>${redisson.version}</version>
</dependency>
```

Bean Configuration

Spring Boot does not provide Redisson adaptation, and the **application.properties** configuration file does not have the corresponding configuration item. Therefore, you can only use Bean configuration.

- Single-node and Proxy Cluster

```
import org.redisson.Redisson;
import org.redisson.api.RedissonClient;
import org.redisson.codec.JsonJacksonCodec;
import org.redisson.config.Config;
import org.redisson.config.SingleServerConfig;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SingleConfig {

    @Value("${redis.address:}")
    private String redisAddress;

    @Value("${redis.password:}")
    private String redisPassword;
    @Value("${redis.database:0}")
    private Integer redisDatabase = 0;

    @Value("${redis.connect.timeout:3000}")
    private Integer redisConnectTimeout = 3000;

    @Value("${redis.connection.idle.timeout:10000}")
    private Integer redisConnectionIdleTimeout = 10000;

    @Value("${redis.connection.ping.interval:1000}")
    private Integer redisConnectionPingInterval = 1000;

    @Value("${redis.timeout:2000}")
    private Integer timeout = 2000;

    @Value("${redis.connection.pool.min.size:50}")
    private Integer redisConnectionPoolMinSize;

    @Value("${redis.connection.pool.max.size:200}")
    private Integer redisConnectionPoolMaxSize;

    @Value("${redis.retry.attempts:3}")
    private Integer redisRetryAttempts = 3;

    @Value("${redis.retry.interval:200}")
    private Integer redisRetryInterval = 200;

    @Bean
    public RedissonClient redissonClient(){
        Config redissonConfig = new Config();

        SingleServerConfig serverConfig = redissonConfig.useSingleServer();
        serverConfig.setAddress(redisAddress);
        serverConfig.setConnectionMinimumIdleSize(redisConnectionPoolMinSize);
        serverConfig.setConnectionPoolSize(redisConnectionPoolMaxSize);

        serverConfig.setDatabase(redisDatabase);
        serverConfig.setPassword(redisPassword);
        serverConfig.setConnectTimeout(redisConnectTimeout);
        serverConfig.setIdleConnectionTimeout(redisConnectionIdleTimeout);
        serverConfig.setPingConnectionInterval(redisConnectionPingInterval);
        serverConfig.setTimeout(timeout);
        serverConfig.setRetryAttempts(redisRetryAttempts);
        serverConfig.setRetryInterval(redisRetryInterval);

        redissonConfig.setCodec(new JsonJacksonCodec());
        return Redisson.create(redissonConfig);
    }
}
```

- Master/Standby

```
import org.redisson.Redisson;
import org.redisson.api.RedissonClient;
import org.redisson.codec.JsonJacksonCodec;
import org.redisson.config.Config;
import org.redisson.config.MasterSlaveServersConfig;
import org.redisson.config.ReadMode;
import org.redisson.config.SubscriptionMode;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.HashSet;

@Configuration
public class MasterStandbyConfig {
    @Value("${redis.master.address}")
    private String redisMasterAddress;

    @Value("${redis.slave.address}")
    private String redisSlaveAddress;

    @Value("${redis.database:0}")
    private Integer redisDatabase = 0;

    @Value("${redis.password:}")
    private String redisPassword;

    @Value("${redis.connect.timeout:3000}")
    private Integer redisConnectTimeout = 3000;

    @Value("${redis.connection.idle.timeout:10000}")
    private Integer redisConnectionIdleTimeout = 10000;

    @Value("${redis.connection.ping.interval:1000}")
    private Integer redisConnectionPingInterval = 1000;

    @Value("${redis.timeout:2000}")
    private Integer timeout = 2000;

    @Value("${redis.master.connection.pool.min.size:50}")
    private Integer redisMasterConnectionPoolMinSize = 50;

    @Value("${redis.master.connection.pool.max.size:200}")
    private Integer redisMasterConnectionPoolMaxSize = 200;

    @Value("${redis.retry.attempts:3}")
    private Integer redisRetryAttempts = 3;

    @Value("${redis.retry.interval:200}")
    private Integer redisRetryInterval = 200;

    @Bean
    public RedissonClient redissonClient() {
        Config redissonConfig = new Config();

        MasterSlaveServersConfig serverConfig = redissonConfig.useMasterSlaveServers();
        serverConfig.setMasterAddress(redisMasterAddress);
        HashSet<String> slaveSet = new HashSet<>();
        slaveSet.add(redisSlaveAddress);
        serverConfig.setSlaveAddresses(slaveSet);

        serverConfig.setDatabase(redisDatabase);
        serverConfig.setPassword(redisPassword);

        serverConfig.setMasterConnectionMinimumIdleSize(redisMasterConnectionPoolMinSize);
        serverConfig.setMasterConnectionPoolSize(redisMasterConnectionPoolMaxSize);

        serverConfig.setReadMode(ReadMode.MASTER_SLAVE);
        serverConfig.setSubscriptionMode(SubscriptionMode.MASTER);

        serverConfig.setConnectTimeout(redisConnectTimeout);
        serverConfig.setIdleConnectionTimeout(redisConnectionIdleTimeout);
        serverConfig.setPingConnectionInterval(redisConnectionPingInterval);
        serverConfig.setTimeout(timeout);
        serverConfig.setRetryAttempts(redisRetryAttempts);
        serverConfig.setRetryInterval(redisRetryInterval);

        redissonConfig.setCodec(new JsonJacksonCodec());
        return Redisson.create(redissonConfig);
    }
}
```

- Redis Cluster

```
import org.redisson.Redisson;
import org.redisson.api.RedissonClient;
import org.redisson.codec.JsonJacksonCodec;
import org.redisson.config.ClusterServersConfig;
import org.redisson.config.Config;
import org.redisson.config.ReadMode;
import org.redisson.config.SubscriptionMode;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.List;

@Configuration
public class ClusterConfig {

    @Value("${redis.cluster.address}")
    private List<String> redisClusterAddress;

    @Value("${redis.cluster.scan.interval:5000}")
    private Integer redisClusterScanInterval = 5000;

    @Value("${redis.password:}")
    private String redisPassword;

    @Value("${redis.connect.timeout:3000}")
    private Integer redisConnectTimeout = 3000;

    @Value("${redis.connection.idle.timeout:10000}")
    private Integer redisConnectionIdleTimeout = 10000;

    @Value("${redis.connection.ping.interval:1000}")
    private Integer redisConnectionPingInterval = 1000;

    @Value("${redis.timeout:2000}")
    private Integer timeout = 2000;

    @Value("${redis.retry.attempts:3}")
    private Integer redisRetryAttempts = 3;

    @Value("${redis.retry.interval:200}")
    private Integer redisRetryInterval = 200;

    @Value("${redis.master.connection.pool.min.size:50}")
    private Integer redisMasterConnectionPoolMinSize = 50;

    @Value("${redis.master.connection.pool.max.size:200}")
    private Integer redisMasterConnectionPoolMaxSize = 200;

    @Bean
    public RedissonClient redissonClient() {
        Config redissonConfig = new Config();

        ClusterServersConfig serverConfig = redissonConfig.useClusterServers();
        serverConfig.setNodeAddresses(redisClusterAddress);
        serverConfig.setScanInterval(redisClusterScanInterval);

        serverConfig.setPassword(redisPassword);

        serverConfig.setMasterConnectionMinimumIdleSize(redisMasterConnectionPoolMinSize);
        serverConfig.setMasterConnectionPoolSize(redisMasterConnectionPoolMaxSize);

        serverConfig.setReadMode(ReadMode.MASTER);
        serverConfig.setSubscriptionMode(SubscriptionMode.MASTER);

        serverConfig.setConnectTimeout(redisConnectTimeout);
        serverConfig.setIdleConnectionTimeout(redisConnectionIdleTimeout);
        serverConfig.setPingConnectionInterval(redisConnectionPingInterval);
        serverConfig.setTimeout(timeout);
        serverConfig.setRetryAttempts(redisRetryAttempts);
        serverConfig.setRetryInterval(redisRetryInterval);

        redissonConfig.setCodec(new JsonJacksonCodec());
        return Redisson.create(redissonConfig);
    }
}
```

(Optional) Configuring SSL Connections

If SSL is enabled for an instance, to access it using SSL connections, add the **configRedissonSSL(serverConfig)** logic to the **RedissonClient** construction method **clientConfiguration()** in Bean Configuration and change the Redis addresses from **redis://*ip*:*port*** to **rediss://*ip*:*port***. For details about whether your DCS Redis instances support SSL, see **Transmitting DCS Redis Data with Encryption Using SSL**.

```
private void configRedissonSSL(BaseConfig serverConfig) {
    TrustManagerFactory trustManagerFactory = null;
    try {
        //Load the CA certificate in the user-defined path.
        CertificateFactory cf = CertificateFactory.getInstance("X.509");
        Certificate ca;
        try (InputStream is = new FileInputStream(certificationPath)) {
            ca = cf.generateCertificate(is);
        }

        //Create keystore.
        String keyStoreType = KeyStore.getDefaultType();
        KeyStore keyStore = KeyStore.getInstance(keyStoreType);
        keyStore.load(null, null);
        keyStore.setCertificateEntry("ca", ca);

        //Create TrustManager.
        trustManagerFactory = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
        trustManagerFactory.init(keyStore);
    } catch (CertificateException | IOException | KeyStoreException | NoSuchAlgorithmException e) {
        e.printStackTrace();
        return;
    }

    serverConfig.setSslTrustManagerFactory(trustManagerFactory);
}
```

Parameter Description

Table 4-14 Config parameters

Parameter	Default Value	Description
codec	org.redisson.codec.JsonJacksonCodec	Encoding format, including JSON, Avro, Smile, CBOR, and MsgPack.
threads	Number of CPU cores x 2	Thread pool used for executing RTopic Listener, RRemoteService, and RExecutorService.
executor	null	The function is the same as **threads**. If this parameter is not set, a thread pool is initialized based on **threads**.

Parameter	Default Value	Description
nettyThreads	Number of CPU cores x 2	Thread pool used by the TCP channel that connects to the redis-server. All channels share this connection pool and are mapped to Netty's **Bootstrap.group(...)**.
eventLoopGroup	null	The function is the same as **nettyThreads**. If this parameter is not set, an EventLoopGroup is initialized based on the **nettyThreads** parameter for the bottom-layer TCP channel to use.
transportMode	TransportMode.NIO	Transmission mode. The options are **NIO**, **EPOLL** (additional package required), and **KQUEUE** (additional package required).
lockWatchdogTimeout	30000	Timeout interval (in milliseconds) of the lock-monitoring watchdog. In the distributed lock scenario, if the **leaseTimeout** parameter is not specified, the default value of this parameter is used.
keepPubSubOrder	true	Indicates whether to receive messages in the publish sequence. **If messages can be processed concurrently, you are advised to set this parameter to false.**

Table 4-15 SingleServerConfig parameters (single-node, or Proxy Cluster)

Parameter	Default Value	Description
address	-	Node connection information, in redis://*ip*:*port* format.
database	0	ID of the database to be used.
connectionMinimumIdleSize	32	Minimum number of connections to the master node of each shard.
connectionPoolSize	64	Maximum number of connections to the master node of each shard.
subscriptionConnectionMinimumIdleSize	1	Minimum number of connections to the target node for pub/sub.
subscriptionConnectionPoolSize	50	Maximum number of connections to the target node for pub/sub.
subcriptionPerConnection	5	Maximum number of subscriptions on each subscription connection.

Parameter	Default Value	Description
connectionTimeout	10000	Connection timeout interval, in milliseconds.
idleConnectionTimeout	10000	Maximum time (in milliseconds) for reclaiming idle connections.
pingConnectionInterval	30000	Heartbeat for detecting available connections, in milliseconds. **Recommended: 3000 ms**.
timeout	3000	Timeout interval for waiting for a response, in milliseconds.
retryAttempts	3	Maximum number of retries upon send failures.
retryInterval	1500	Retry interval, in milliseconds. **Recommended: 200 ms**.
clientName	null	Client name.

Table 4-16 MasterSlaveServersConfig parameters (master/standby)

Parameter	Default Value	Description
masterAddress	-	Master node connection information, in redis://*ip:port* format.
slaveAddresses	-	Standby node connection information, in Set<redis://*ip:port*> format.
readMode	SLAVE	Read mode. By default, read traffic is distributed to replica nodes. The value can be **MASTER** (recommended), **SLAVE**, or **MASTER_SLAVE**.
loadBalancer	RoundRobinLoadBalancer	Load balancing algorithm. This parameter is valid only when **readMode** is set to **SLAVE** or **MASTER_SLAVE**. Read traffic is distributed evenly.
masterConnectionMinimumIdleSize	32	Minimum number of connections to the master node of each shard.
masterConnectionPoolSize	64	Maximum number of connections to the master node of each shard.
slaveConnectionMinimumIdleSize	32	Minimum number of connections to each replica node of each shard. If **readMode** is set to **MASTER**, the value of this parameter is invalid.

Parameter	Default Value	Description
slaveConnectionPoolSize	64	Maximum number of connections to each replica node of each shard. If **readMode** is set to **MASTER**, the value of this parameter is invalid.
subscriptionMode	SLAVE	Subscription mode. By default, only replica nodes handle subscription. The value can be **SLAVE** or **MASTER** (recommended).
subscriptionConnectionMinimumIdleSize	1	Minimum number of connections to the target node for pub/sub.
subscriptionConnectionPoolSize	50	Maximum number of connections to the target node for pub/sub.
subcriptionPerConnection	5	Maximum number of subscriptions on each subscription connection.
connectionTimeout	10000	Connection timeout interval, in milliseconds.
idleConnectionTimeout	10000	Maximum time (in milliseconds) for reclaiming idle connections.
pingConnectionInterval	30000	Heartbeat for detecting available connections, in milliseconds. **Recommended: 3000 ms**.
timeout	3000	Timeout interval for waiting for a response, in milliseconds.
retryAttempts	3	Maximum number of retries upon send failures.
retryInterval	1500	Retry interval, in milliseconds. **Recommended: 200 ms**.
clientName	null	Client name.

Table 4-17 ClusterServersConfig parameters (Redis Cluster)

Parameter	Default Value	Description
nodeAddress	-	Connection addresses of cluster nodes. Each address uses the redis://*ip:port* format. Use commas (,) to separate connection addresses of different nodes.
password	null	Password for logging in to the cluster.

Parameter	Default Value	Description
scanInterval	1000	Interval for periodically checking the cluster node status, in milliseconds.
readMode	SLAVE	Read mode. By default, read traffic is distributed to replica nodes. The value can be **MASTER** (recommended), **SLAVE**, or **MASTER_SLAVE**.
loadBalancer	RoundRobinLoadBalancer	Load balancing algorithm. This parameter is valid only when **readMode** is set to **SLAVE** or **MASTER_SLAVE**. Read traffic is distributed evenly.
masterConnectionMinimumIdleSize	32	Minimum number of connections to the master node of each shard.
masterConnectionPoolSize	64	Maximum number of connections to the master node of each shard.
slaveConnectionMinimumIdleSize	32	Minimum number of connections to each replica node of each shard. If **readMode** is set to **MASTER**, the value of this parameter is invalid.
slaveConnectionPoolSize	64	Maximum number of connections to each replica node of each shard. If **readMode** is set to **MASTER**, the value of this parameter is invalid.
subscriptionMode	SLAVE	Subscription mode. By default, only replica nodes handle subscription. The value can be **SLAVE** or **MASTER** (recommended).
subscriptionConnectionMinimumIdleSize	1	Minimum number of connections to the target node for pub/sub.
subscriptionConnectionPoolSize	50	Maximum number of connections to the target node for pub/sub.
subcriptionPerConnection	5	Maximum number of subscriptions on each subscription connection.
connectionTimeout	10000	Connection timeout interval, in milliseconds.
idleConnectionTimeout	10000	Maximum time (in milliseconds) for reclaiming idle connections.
pingConnectionInterval	30000	Heartbeat for detecting available connections, in milliseconds. **Recommended: 3000**.

Parameter	Default Value	Description
timeout	3000	Timeout interval for waiting for a response, in milliseconds.
retryAttempts	3	Maximum number of retries upon send failures.
retryInterval	1500	Retry interval, in milliseconds. **Recommended: 200**.
clientName	null	Client name.

Suggestion for Configuring DCS Instances

- **readMode**

  MASTER is the recommended value, that is, the master node bears all read and write traffic. This is to avoid data inconsistency caused by master/replica synchronization latency. If the value is **SLAVE**, all read requests will trigger errors when replicas are faulty. If the value is **MASTER_SLAVE**, some read requests will trigger errors. Read errors last for the period specified by **failedSlaveCheckInterval** (180s by default) until the faulty nodes are removed from the available node list.

- **subscriptionMode**

  Similarly, **MASTER** is the recommended value.

- Connection pool configuration

  NOTE

  The following calculation is applicable only to common service scenarios. You can customize it based on your service requirements.

  There is no standard connection pool size. You can configure one based on your service traffic. The following formulas are for reference:

  - Minimum number of connections = (QPS of a single node accessing Redis)/(1000 ms/Average time spent on a single command)
  - Maximum number of connections = (QPS of a single node accessing Redis)/(1000 ms/Average time spent on a single command) x 150%

  For example, if the QPS of a service application is about 10,000, each request needs to access Redis 10 times (that is, 100,000 accesses to Redis every second), and the service application has 10 hosts, the calculation is as follows:

  QPS of a single node accessing Redis = 100,000/10 = 10,000

  Average time spent on a single command = 20 ms (Redis takes 5 ms to 10 ms to process a single command under normal conditions. If network jitter occurs, it takes 15 ms to 20 ms.)

  Minimum number of connections = 10,000/(1000 ms/20 ms) = 200

  Maximum number of connections = 10,000/(1000 ms/20 ms) x 150% = 300

- Retry configuration

Redisson supports retries. You can set the following parameters based on service requirements. Generally, configure three retries, and set the retry interval to about 200 ms.

- **retryAttempts**: number of retry times
- **retryInterval**: retry interval

NOTE

In Redisson, some APIs are implemented through LUA, and the performance is low. You are advised to use Jedis instead of Redisson.

4.2.3.2 Connecting to Redis on redis-py (Python)

This section describes how to access a Redis instance on redis-py. For more information about how to use other Redis clients, visit **the Redis official website**.

The following operations are based on an example of accessing a Redis instance on a client on an elastic cloud server (ECS).

NOTE

Use redis-py to connect to single-node, master/standby, and Proxy Cluster instances and redis-py-cluster to connect to Redis Cluster instances.

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see the *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the Python compilation environment has been installed on the ECS.

Connecting to Redis on redis-py

Step 1 View the IP address and port number of the DCS Redis instance to be accessed.

For details, see **Viewing Details of a DCS Instance**.

Step 2 Log in to the ECS.

The following uses CentOS as an example to describe how to access an instance using a Python client.

Step 3 Access the DCS Redis instance.

If the system does not provide Python, run the following **yum** command to install it:

yum install python

NOTE

The Python version must be 3.6 or later. If the default Python version is earlier than 3.6, perform the following operations to change it:

1. Run the **rm -rf python** command to delete the Python symbolic link.
2. Run the **ln -s python***X.X.X* **python** command to create another Python link. In the command, *X.X.X* indicates the Python version number.

- If the instance is a single-node, master/standby, or Proxy Cluster instance:
  a. Install Python and redis-py.
     i. If the system does not provide Python, run the **yum** command to install it.
     ii. Run the following command to download and decompress the redis-py package:

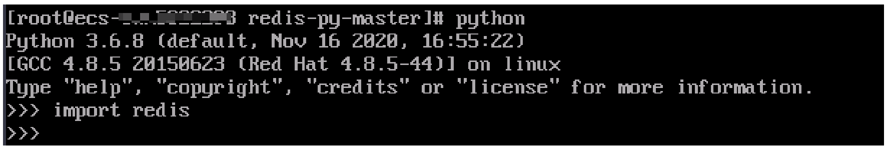
        wget https://github.com/andymccurdy/redis-py/archive/master.zip

        unzip master.zip
     iii. Go to the directory where the decompressed redis-py package is saved, and install redis-py.

        python setup.py install

        After the installation, run the **python** command. redis-py have been successfully installed if the following command output is displayed:

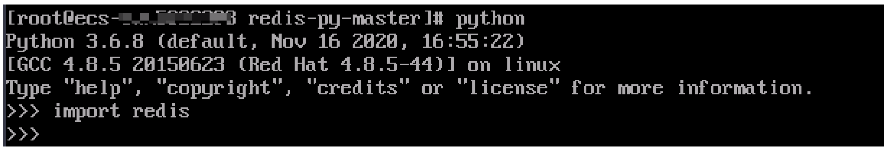
Figure 4-2 Running the python command



```
[root@ecs-[redacted] redis-py-master]# python
Python 3.6.8 (default, Nov 16 2020, 16:55:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import redis
>>>
```

  b. Use the redis-py client to connect to the instance. In the following steps, commands are executed in CLI mode. (Alternatively, write the commands into a Python script and then execute the script.)
     i. Run the **python** command to enter the CLI mode. You have entered CLI mode if the following command output is displayed:

Figure 4-3 Entering the CLI mode



```
[root@ecs-[redacted] redis-py-master]# python
Python 3.6.8 (default, Nov 16 2020, 16:55:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import redis
>>>
```

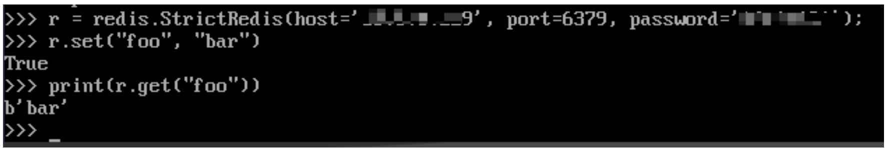
     ii. Run the following command to access the chosen DCS Redis instance:

```
r = redis.StrictRedis(host='XXX.XXX.XXX.XXX', port=6379, password='******');
```

        XXX.XXX.XXX.XXX indicates the IP address of the DCS instance and **6379** is an example port number of the instance. For details about how to obtain the IP address and port, see **Step 1**. Change them as required. ******** indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

        You have successfully accessed the instance if the following command output is displayed. Enter commands to perform read and write operations on the database.

Figure 4-4 Redis connected successfully



```
>>> r = redis.StrictRedis(host='[redacted]', port=6379, password='[redacted]');
>>> r.set("foo", "bar")
True
>>> print(r.get("foo"))
b'bar'
>>> _
```

- If the instance is a Redis Cluster instance:
  a. Install the redis-py-cluster client.
     i. Download the released version.
        wget https://github.com/Grokzen/redis-py-cluster/releases/download/2.1.3/redis-py-cluster-2.1.3.tar.gz
     ii. Decompress the package.
        tar -xvf redis-py-cluster-2.1.3.tar.gz
     iii. Go to the directory where the decompressed redis-py-cluster package is saved, and install redis-py-cluster.
        python setup.py install
  b. Access the DCS Redis instance by using redis-py-cluster.
     In the following steps, commands are executed in CLI mode. (Alternatively, write the commands into a Python script and then execute the script.)
     i. Run the **python** command to enter the CLI mode.
     ii. Run the following command to access the chosen DCS Redis instance. If the instance does not have a password, exclude **password='******'** from the command.

```
>>> from rediscluster import RedisCluster

>>> startup_nodes = [{"host": "192.168.0.143", "port": "6379"},{"host": "192.168.0.144", "port": "6379"},{"host": "192.168.0.145", "port": "6379"},{"host": "192.168.0.146", "port": "6379"}]

>>> rc = RedisCluster(startup_nodes=startup_nodes, decode_responses=True, password='******')

>>> rc.set("foo", "bar")
True
>>> print(rc.get("foo"))
'bar'
```

----End

4.2.3.3 Connecting to Redis on go-redis (Go)

This section describes how to access a Redis instance on go-redis. For more information about how to use other Redis clients, visit **the Redis official website**.

The following operations are based on an example of accessing a Redis instance on a client on an elastic cloud server (ECS).

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- View the IP address and port number of the DCS Redis instance to be accessed.

For details, see **Viewing Details of a DCS Instance**.

- An ECS has been created. For details about how to create an ECS, see the *Elastic Cloud Server User Guide*.

Connecting to Redis on go-redis

Step 1 Log in to the ECS.

A Windows ECS is used as an example.

Step 2 Install Visual Studio Community 2017 on the ECS.

Step 3 Start Visual Studio and create a project. The project name can be customized. In this example, the project name is set to **redisdemo**.

Step 4 Import the dependency package of go-redis and enter **go get github.com/go-redis/redis** on the terminal.

Step 5 Write the following code:

```
package main

import (
    "fmt"
    "github.com/go-redis/redis"
)

func main() {
    // Single-node
    rdb := redis.NewClient(&redis.Options{
        Addr:     "host:port",
        Password: "********", // no password set
        DB:       0,  // use default DB
    })

    val, err := rdb.Get("key").Result()
    if err != nil {
        if err == redis.Nil {
            fmt.Println("key does not exists")
            return
        }
        panic(err)
    }
    fmt.Println(val)

    //Cluster
    rdbCluster := redis.NewClusterClient(&redis.ClusterOptions{
        Addrs:    []string{"host:port"},
        Password: "********",
    })
    val1, err1 := rdbCluster.Get("key").Result()
    if err1 != nil {
        if err == redis.Nil {
            fmt.Println("key does not exists")
            return
        }
        panic(err)
    }
    fmt.Println(val1)
}
```

host:port are the IP address and port number of the DCS Redis instance. For details about how to obtain the IP address and port, see **Prerequisites**. Change them as required. ********** indicates the password used to log in to the DCS Redis instance. This password is defined during DCS Redis instance creation.

Step 6 Run the **go build -o test main.go** command to package the code into an executable file, for example, **test**.

> ⚠ CAUTION
>
> To run the package in the Linux OS, set the following parameters before packaging:
>
> **set GOARCH=amd64**
>
> **set GOOS=linux**

Step 7 Run the **./test** command to access the DCS instance.

----End

4.2.3.4 Connecting to Redis on hiredis (C++)

This section describes how to access a Redis instance on hiredis. For more information about how to use other Redis clients, visit **the Redis official website**.

The following operations are based on an example of accessing a Redis instance on a client on an elastic cloud server (ECS).

> NOTE
>
> The operations described in this section apply only to single-node, master/standby, and Proxy Cluster instances. To use C++ to connect to a Redis Cluster instance, see the **C++ Redis client description**.

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see the *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the GCC compilation environment has been installed on the ECS.

Connecting to Redis on hiredis

Step 1 View the IP address and port number of the DCS Redis instance to be accessed.

For details, see **Viewing Details of a DCS Instance**.

Step 2 Log in to the ECS.

The following uses CentOS as an example to describe how to access an instance in C++.

Step 3 Install GCC, Make, and hiredis.

If the system does not provide a compiling environment, run the following **yum** command to install the environment:

yum install gcc make

Step 4 Run the following command to download and decompress the hiredis package:

wget https://github.com/redis/hiredis/archive/master.zip

unzip master.zip

Step 5 Go to the directory where the decompressed hiredis package is saved, and compile and install hiredis.

make

make install

Step 6 Access the DCS instance by using hiredis.

The following describes connection and password authentication of hiredis. For more information on how to use hiredis, visit the Redis official website.

1. Edit the sample code for connecting to a DCS instance, and then save the code and exit.

   vim connRedis.c

   Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <hiredis.h>
int main(int argc, char **argv) {
    unsigned int j;
    redisContext *conn;
    redisReply *reply;
    if (argc < 3) {
            printf("Usage: example {instance_ip_address} 6379 {password}\n");
            exit(0);
    }
    const char *hostname = argv[1];
    const int port = atoi(argv[2]);
    const char *password = argv[3];
    struct timeval timeout = { 1, 500000 }; // 1.5 seconds
    conn = redisConnectWithTimeout(hostname, port, timeout);
    if (conn == NULL || conn->err) {
       if (conn) {
            printf("Connection error: %s\n", conn->errstr);
            redisFree(conn);
       } else {
            printf("Connection error: can't allocate redis context\n");
       }
    exit(1);
    }
    /* AUTH */
    reply = redisCommand(conn, "AUTH %s", password);
    printf("AUTH: %s\n", reply->str);
    freeReplyObject(reply);

    /* Set */
    reply = redisCommand(conn,"SET %s %s", "welcome", "Hello, DCS for Redis!");
    printf("SET: %s\n", reply->str);
    freeReplyObject(reply);

    /* Get */
    reply = redisCommand(conn,"GET welcome");
    printf("GET welcome: %s\n", reply->str);
    freeReplyObject(reply);

    /* Disconnects and frees the context */
    redisFree(conn);
    return 0;
}
```

2. Run the following command to compile the code:

   ```
   gcc connRedis.c -o connRedis -I /usr/local/include/hiredis -lhiredis
   ```

   If an error is reported, locate the directory where the **hiredis.h** file is saved and modify the compilation command.

   After the compilation, an executable **connRedis** file is obtained.

3. Run the following command to access the chosen DCS Redis instance:

   ./connRedis *{redis_instance_address}* 6379 *{password}*

   {redis_instance_address} indicates the IP address of DCS instance and **6379** is an example port number of DCS instance. For details about how to obtain the IP address and port, see **Step 1**. Change them as required. *{password}* indicates the password used to log in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

   You have successfully accessed the instance if the following command output is displayed:

   ```
   AUTH: OK
   SET: OK
   GET welcome: Hello, DCS for Redis!
   ```

NOTICE

If an error is reported, indicating that the hiredis library files cannot be found, run the following commands to copy related files to the system directories and add dynamic links:

```
mkdir /usr/lib/hiredis
cp /usr/local/lib/libhiredis.so.0.13 /usr/lib/hiredis/
mkdir /usr/include/hiredis
cp /usr/local/include/hiredis/hiredis.h /usr/include/hiredis/
echo '/usr/local/lib' >>;>>;/etc/ld.so.conf
ldconfig
```

Replace the locations of the **so** and **.h** files with actual ones before running the commands.

----End

4.2.3.5 Connecting to Redis on StackExchange.Redis (C#)

This section describes how to access a Redis instance on StackExchange.Redis. For more information about how to use other Redis clients, visit **the Redis official website**.

The following operations are based on an example of accessing a Redis instance on a client on an elastic cloud server (ECS).

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.

- An ECS has been created. For details about how to create an ECS, see the *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the GCC compilation environment has been installed on the ECS.

Connecting to Redis on StackExchange.Redis

Step 1 View the IP address and port number of the DCS Redis instance to be accessed.

For details, see **Viewing Details of a DCS Instance**.

Step 2 Log in to the ECS.

A Windows ECS is used as an example.

Step 3 Install Visual Studio Community 2017 on the ECS.

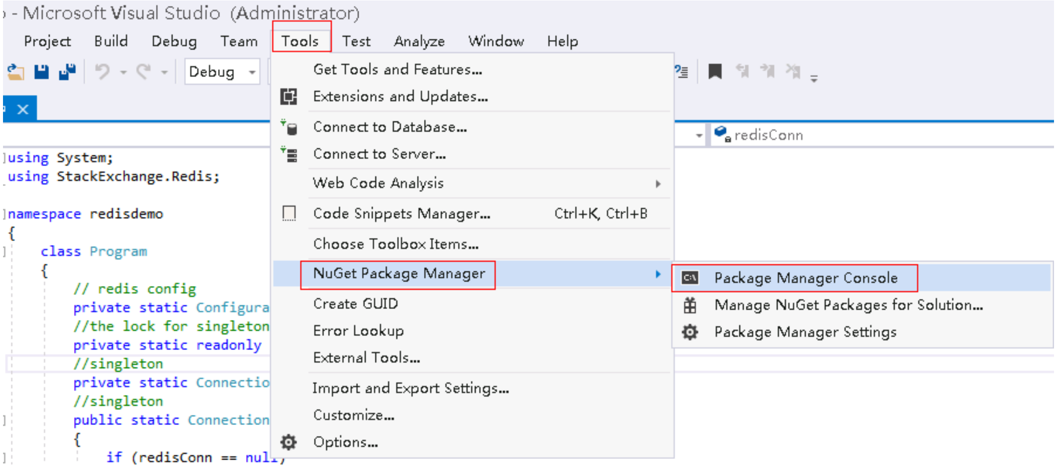
Step 4 Start Visual Studio 2017 and create a project.

Set the project name to **redisdemo**.

Step 5 Install StackExchange.Redis by using the NuGet package manager of Visual Studio.

Access the NuGet package manager console according to **Figure 4-5**, and enter **Install-Package StackExchange.Redis -*Version 2.2.79*.** (The version number is optional).

Figure 4-5 Accessing the NuGet package manager console



```
using System;
using StackExchange.Redis;

namespace redisdemo
{
    class Program
    {
        // redis config
        private static Configura
        //the lock for singleton
        private static readonly
        //singleton
        private static Connectio
        //singleton
        public static Connection
        {
            if (redisConn == null)
```

Step 6 Write the following code, and use the String Set and Get methods to test the connection.

```
using System;
using StackExchange.Redis;

namespace redisdemo
{
    class Program
    {
        // redis config
        private static ConfigurationOptions connDCS = ConfigurationOptions.Parse("{instance_ip_address}:
{port},password=********,connectTimeout=2000");
        //the lock for singleton
        private static readonly object Locker = new object();
        //singleton
        private static ConnectionMultiplexer redisConn;
        //singleton
        public static ConnectionMultiplexer getRedisConn()
        {
            if (redisConn == null)
            {
                lock (Locker)
                {
                    if (redisConn == null || !redisConn.IsConnected)
                    {
                        redisConn = ConnectionMultiplexer.Connect(connDCS);
                    }
                }
            }
            return redisConn;
        }
        static void Main(string[] args)
        {
            redisConn = getRedisConn();
            var db = redisConn.GetDatabase();
            //set get
            string strKey = "Hello";
            string strValue = "DCS for Redis!";
            Console.WriteLine( strKey + ", " + db.StringGet(strKey));

            Console.ReadLine();
        }
    }
}
```

{instance_ip_address} and *{port}* are the IP address and port number of the DCS Redis instance. For details about how to obtain the IP address and port, see **Step 1**. Change them as required. ********** indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

Step 7 Run the code. You have successfully accessed the instance if the following command output is displayed:

```
Hello, DCS for Redis!
```

For more information about other commands of StackExchange.Redis, visit **StackExchange.Redis**.

----End

4.2.3.6 PHP

4.2.3.6.1 Connecting to Redis on phpredis (PHP)

This section describes how to connect to Redis on phpredis. For more information about how to use other Redis clients, visit **the Redis official website**.

NOTE

The operations described in this section apply only to single-node, master/standby, and Proxy Cluster instances. To use phpredis to connect to a Redis Cluster instance, see the **phpredis description**.

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.

- An ECS has been created. For details about how to create an ECS, see the *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the GCC compilation environment has been installed on the ECS.

Connecting to Redis on phpredis

Step 1 View the IP address and port number of the DCS Redis instance to be accessed.

For details, see **Viewing Details of a DCS Instance**.

Step 2 Log in to the ECS.

The following uses CentOS as an example to describe how to access an instance through phpredis.

Step 3 Install GCC-C++ and Make compilation components.

yum install gcc-c++ make

Step 4 Install the PHP development package and CLI tool.

Run the following **yum** command to install the PHP development package:

yum install php-devel php-common php-cli

After the installation is complete, run the following command to query the PHP version and check whether the installation is successful:

php --version

Step 5 Install the phpredis client.

1. Download the source phpredis package.

   wget http://pecl.php.net/get/redis-5.3.7.tgz

   This version is used as an example. To download phpredis clients of other versions, visit the Redis or PHP official website.
2. Decompress the source phpredis package.

   tar -zxvf redis-5.3.7.tgz

   cd redis-5.3.7
3. Command before compilation.

   phpize
4. Configure the **php-config** file.

   ./configure --with-php-config=/usr/bin/php-config

   The location of the file varies depending on the OS and PHP installation mode. You are advised to locate the directory where the file is saved before the configuration.

   find / -name php-config
5. Compile and install the phpredis client.

   make && make install
6. After the installation, add the **extension** configuration in the **php.ini** file to reference the Redis module.

   vim /etc/php.ini

Add the following configuration:

```
extension = "/usr/lib64/php/modules/redis.so"
```

NOTE

The **redis.so** file may be saved in a different directory from **php.ini**. Run the following command to locate the directory:

find / -name php.ini

7. Save the configuration and exit. Then, run the following command to check whether the extension takes effect:

   php -m |grep redis

   If the command output contains **redis**, the phpredis client environment has been set up.

Step 6 Access the DCS instance by using phpredis.

1. Edit a **redis.php** file.

```
<?php
    $redis_host = "{redis_instance_address}";
    $redis_port = {port};
    $user_pwd = "{password}";
    $redis = new Redis();
    if ($redis->connect($redis_host, $redis_port) == false) {
       die($redis->getLastError());
    }
    if ($redis->auth($user_pwd) == false) {
        die($redis->getLastError());
    }
    if ($redis->set("welcome", "Hello, DCS for Redis!") == false) {
        die($redis->getLastError());
    }
    $value = $redis->get("welcome");
    echo $value;
    $redis->close();
?>
```

   {redis_instance_address} indicates the example IP address of the DCS instance and *{port}* indicates the port number of the DCS instance. For details about how to obtain the IP address and port, see **Step 1**. Change them as required. *{password}* indicates the password used to log in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation. If password-free access is enabled, shield the **if** statement for password authentication.

2. Run the **php redis.php** command to access the DCS instance.

----End

4.2.3.6.2 Connecting to Redis on predis (PHP)

This section describes how to connect to Redis on predis. For more information about how to use other Redis clients, visit **the Redis official website**.

Prerequisites

- A DCS Redis instance has been created, and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see the *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the PHP compilation environment has been installed on the ECS.

Connecting to Redis on predis

Step 1 View the IP address and port number of the DCS Redis instance to be accessed.

For details, see **Viewing Details of a DCS Instance**.

Step 2 Log in to the ECS.

Step 3 Install the PHP development package and CLI tool. Run the following **yum** command:

yum install php-devel php-common php-cli

Step 4 After the installation is complete, check the version number to ensure that the installation is successful.

php --version

Step 5 Download the Predis package to the **/usr/share/php** directory.

1. Run the following command to download the Predis source file:

   wget https://github.com/predis/predis/archive/refs/tags/v2.2.2.tar.gz

   NOTE

   This version is used as an example. To download Predis clients of other versions, visit the Redis or PHP official website.

2. Run the following commands to decompress the source Predis package:

   tar -zxvf predis-2.2.2.tar.gz

3. Rename the decompressed Predis directory **predis** and move it to **/usr/share/php/**.

   mv predis-2.2.2 predis

Step 6 Edit a file used to connect to Redis.

- Example of using **redis.php** to connect to a single-node, master/standby, or Proxy Cluster DCS Redis instance:

```
<?php
    require 'predis/autoload.php';
    Predis\Autoloader::register();
    $client = new Predis\Client([
      'scheme' => 'tcp' ,
      'host'     => '{redis_instance_address}' ,
      'port'     =>{port} ,
      'password' => '{password}'
    ]);
    $client->set('foo', 'bar');
    $value = $client->get('foo');
    echo $value;
?>
```

- Example code for using **redis-cluster.php** to connect to Redis Cluster:

```
<?php
   require 'predis/autoload.php';
        $servers = array(
         'tcp://{redis_instance_address}:{port}'
        );
       $options = array('cluster' => 'redis');
       $client = new Predis\Client($servers, $options);
       $client->set('foo', 'bar');
       $value = $client->get('foo');
       echo $value;
?>
```

{redis_instance_address} indicates the actual IP address of the DCS instance and *{port}* is the actual port number of DCS instance. For details about how to obtain the IP address and port, see **Step 1**. Change them as required. *{password}* indicates the password used to log in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation. If password-free access is required, delete the line that contains "password".

Step 7 Run the **php redis.php** command to access the DCS instance.

----End

4.2.3.7 Connecting to Redis on ioredis (Node.js)

This section describes how to access a Redis instance on ioredis. For more information about how to use other Redis clients, visit **the Redis official website**.

NOTE

The operations described in this section apply only to single-node, master/standby, and Proxy Cluster instances. To access a Redis Cluster instance on ioredis, see **Node.js Redis client description**.

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see the *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the GCC compilation environment has been installed on the ECS.

Connecting to Redis on ioredis

- **For client servers running Ubuntu (Debian series):**

Step 1 View the IP address and port number of the DCS Redis instance to be accessed.

For details, see **Viewing Details of a DCS Instance**.

Step 2 Log in to the ECS.

Step 3 Install Node.js.

apt install nodejs-legacy

If the preceding command does not work, run the following commands:

wget https://nodejs.org/dist/v0.12.4/node-v0.12.4.tar.gz --no-check-certificate

tar -xvf node-v4.28.5.tar.gz

cd node-v4.28.5

./configure

make

make install

NOTE

After the installation is complete, run the **node --version** command to query the Node.js version to check whether the installation is successful.

Step 4 Install the node package manager (npm).

apt install npm

Step 5 Install the Redis client ioredis.

npm install ioredis

Step 6 Edit the sample script for connecting to a DCS instance.

Add the following content to the **ioredisdemo.js** script, including information about connection and data reading.

```
var Redis = require('ioredis');
var redis = new Redis({
  port: 6379,          // Redis port
  host: '192.168.0.196',   // Redis host
  family: 4,           // 4 (IPv4) or 6 (IPv6)
  password: '******',
  db: 0
});
redis.set('foo', 'bar');
redis.get('foo', function (err, result) {
  console.log(result);
});
// Or using a promise if the last argument isn't a function
redis.get('foo').then(function (result) {
  console.log(result);
});
// Arguments to commands are flattened, so the following are the same:
redis.sadd('set', 1, 3, 5, 7);
redis.sadd('set', [1, 3, 5, 7]);
// All arguments are passed directly to the redis server:
redis.set('key', 100, 'EX', 10);
```

host indicates the example IP address of the DCS instance and *port* indicates the port number of the DCS instance. For details about how to obtain the IP address and port, see **Step 1**. Change them as required. ******** indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

Step 7 Run the sample script to access the chosen DCS instance.

node ioredisdemo.js

----End

- **For client servers running CentOS (Red Hat series):**

Step 1 View the IP address and port number of the DCS Redis instance to be accessed.

For details, see .

Step 2 Log in to the ECS.

Step 3 Install Node.js.

yum install nodejs

If the preceding command does not work, run the following commands:

wget https://nodejs.org/dist/v0.12.4/node-v0.12.4.tar.gz --no-check-certificate

tar -xvf node-v0.12.4.tar.gz

cd node-v0.12.4

./configure

make

make install

NOTE

After the installation is complete, run the **node --version** command to query the Node.js version to check whether the installation is successful.

Step 4 Install npm.

yum install npm

Step 5 Install the Redis client ioredis.

npm install ioredis

Step 6 Edit the sample script for connecting to a DCS instance.

Add the following content to the **ioredisdemo.js** script, including information about connection and data reading.

```
var Redis = require('ioredis');
var redis = new Redis({
  port: 6379,          // Redis port
  host: '192.168.0.196',   // Redis host
  family: 4,           // 4 (IPv4) or 6 (IPv6)
  password: '******',
  db: 0
});
redis.set('foo', 'bar');
redis.get('foo', function (err, result) {
  console.log(result);
});
// Or using a promise if the last argument isn't a function
redis.get('foo').then(function (result) {
  console.log(result);
});
// Arguments to commands are flattened, so the following are the same:
redis.sadd('set', 1, 3, 5, 7);
redis.sadd('set', [1, 3, 5, 7]);
// All arguments are passed directly to the redis server:
redis.set('key', 100, 'EX', 10);
```

host indicates the example IP address of the DCS instance and *port* indicates the port number of the DCS instance. For details about how to obtain the IP address and port, see **Step 1**. Change them as required. ******** indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

Step 7 Run the sample script to access the chosen DCS instance.

node ioredisdemo.js

----End

4.2.4 Accessing a DCS Redis 4.0/5.0/6.0 Instance on the Console

Access a DCS Redis instance through Web CLI. This function is supported only by DCS Redis 4.0 and later instances, and not by DCS Redis 3.0 instances.

NOTE

- Do not enter sensitive information in Web CLI to avoid disclosure.
- Keys and values cannot contain spaces.
- If the value is empty, **nil** is returned after the **GET** command is executed.

Prerequisites

The instance is in the **Running** state.

Procedure

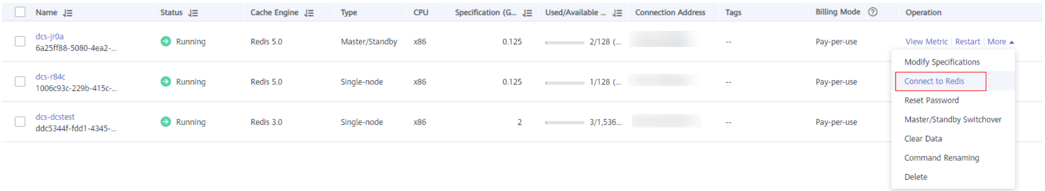
Step 1 Log in to the DCS console.

Step 2 Click in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 In the row containing the desired instance, choose **More** > **Connect to Redis** to go to the Web CLI login page.

Figure 4-6 Connecting to Redis



□	Name	Status	Cache Engine	Type	CPU	Specification (G...	Used/Available ...	Connection Address	Tags	Billing Mode	Operation
□	dcs-jr0a 6a25ff88-5080-4ea2-...	Running	Redis 5.0	Master/Standby	x86	0.125	2/128 (...	[redacted]	--	Pay-per-use	View Metric | Restart | More
□	dcs-r84c 1006c93c-229b-415c-...	Running	Redis 5.0	Single-node	x86	0.125	1/128 (...	[redacted]	--	Pay-per-use	
□	dcs-dcstest ddc5344f-fdd1-4345-...	Running	Redis 3.0	Single-node	x86	2	3/1,536...	[redacted]	--	Pay-per-use	

Step 5 Enter the password of the DCS instance. On Web CLI, select the current Redis database, enter a Redis command in the command box, and press **Enter**.

NOTE

If no operation is performed for more than 5 minutes, the connection times out. You must enter the access password to connect to the instance again.

----End

4.2.5 Accessing a DCS Memcached Instance (Discontinued)

4.2.5.1 Connecting to Memcached on the Telnet

Access a DCS Memcached instance using telnet on an ECS in the same VPC.

Prerequisites

- The DCS Memcached instance you want to access is in the **Running** state.
- An ECS has been created on which the client has been installed. For details on how to create ECSs, see the *Elastic Cloud Server User Guide*.

  NOTE

  An ECS can communicate with a DCS instance that belongs to the same VPC and is configured with the same security group.

  - If the ECS and DCS instance are not in the same VPC, you can establish a VPC peering connection to achieve network connectivity between the ECS and DCS instance. For details, see **Does DCS Support Cross-VPC Access?**
  - If different security groups have been configured for the ECS and DCS instance, you can set security group rules to achieve network connectivity between the ECS and DCS instance. For details, see **Security Group Configurations**.
- All annotations in the example code have been deleted.
- All command lines and code blocks are UTF-8 encoded. Using another encoding scheme will cause compilation problems or even command failures.

Connecting to Memcached on the Telnet

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner of the management console and select the region where your instance is located.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 On the **Cache Manager** page, click the name of the DCS Memcached instance you want to access. Obtain the IP address and port number of the instance.

Step 5 Access the chosen DCS Memcached instance.

1. Log in to the ECS.
2. Run the following command to check whether telnet is installed on the ECS:

   which telnet

   If the directory in which the telnet is installed is displayed, telnet has been installed on the ECS. If the client installation directory is not displayed, install the telnet manually.

   NOTE

   – If telnet has not been installed in Linux, run the **yum -y install telnet** command to install it.

   – In the Windows OS, choose **Start** > **Control Panel** > **Programs** > **Programs and Features** > **Turn Windows features on or off**, and enable telnet.
3. Run the following command to access the chosen DCS Memcached instance:

   telnet *{ip or domain name} {port}*

   In this command: *{ip}* indicates the IP address of the DCS Memcached instance. *{port}* indicates the port number of the DCS Memcached instance. Both the IP address and the port number are obtained in **Step 4**.

   When you have successfully accessed the chosen DCS Memcached instance, information similar to the following is displayed:

```
Trying XXX.XXX.XXX.XXX...
Connected to XXX.XXX.XXX.XXX.
Escape character is '^]'.
```

NOTE

- If **Password Protected** is not enabled for the instance, run the following commands directly after the instance is accessed successfully.
- If **Password Protected** is enabled for the instance, attempts to perform operations on the instance will result in the message "ERROR authentication required", indicating that you do not have the required permissions. In this case, enter **auth *username@password*** to authenticate first. *username* and *password* are that used for accessing the DCS Memcached instance.

Example commands for using the DCS Memcached instance (lines in bold are the commands and the other lines are the command output):

```
set hello 0 0 6
world!
STORED
get hello
VALUE hello 0 6
world!
END
```

----End

4.2.5.2 Connecting to Memcached on the Spymemcached (Java)

Access a DCS Memcached instance using a Java client on an ECS in the same VPC.

Prerequisites

- The DCS Memcached instance you want to access is in the **Running** state.
- An ECS has been created on which the client has been installed. For details on how to create ECSs, see the *Elastic Cloud Server User Guide*.

  NOTE

  An ECS can communicate with a DCS instance that belongs to the same VPC and is configured with the same security group.

  - If the ECS and DCS instance are not in the same VPC, you can establish a VPC peering connection to achieve network connectivity between the ECS and DCS instance. For details, see **Does DCS Support Cross-VPC Access?**
  - If different security groups have been configured for the ECS and DCS instance, you can set security group rules to achieve network connectivity between the ECS and DCS instance. For details, see **Security Group Configurations**.
- The Java development kit (JDK) and common integrated development environments (IDEs) such as Eclipse have been installed on the ECS.
- You have obtained the **spymemcached-*x.y.z*.jar** dependency package.

  NOTE

  x.y.z indicates the version of the dependency package. The latest version is recommended.

Connecting to Memcached on the Spymemcached

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner of the management console and select the region where your instance is located.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 On the **Cache Manager** page, click the name of the DCS Memcached instance you want to access. Obtain the IP address and port number of the instance.

Step 5 Upload the obtained **spymemcached-***x.y.z***.jar** dependency package to the created ECS.

Step 6 Log in to the ECS.

Step 7 Create a Java project on Eclipse and import the **spymemcached-***x.y.z***.jar** dependency package. The project name is customizable.

Step 8 Create a **ConnectMemcached1** class, copy the following Java code to the class, and modify the code.

- Example code for the password mode

  Change *ip:port* to the IP address and port number obtained in **Step 4**. Set *userName* and *password* respectively to the username and password of the Memcached instance.

```
//Connect to the encrypted Memcached code using Java.
import java.io.IOException;
import java.util.concurrent.ExecutionException;

import net.spy.memcached.AddrUtil;
import net.spy.memcached.ConnectionFactoryBuilder;
import net.spy.memcached.ConnectionFactoryBuilder.Protocol;
import net.spy.memcached.MemcachedClient;
import net.spy.memcached.auth.AuthDescriptor;
import net.spy.memcached.auth.PlainCallbackHandler;
import net.spy.memcached.internal.OperationFuture;

public class ConnectMemcached1
{
    public static void main(String[] args)
    {
        final String connectionaddress = "ip:port";
        final String username = "userName";//Indicates the username.
        final String password = "password";//Indicates the password.
        MemcachedClient client = null;
        try
        {
            AuthDescriptor authDescriptor =
                new AuthDescriptor(new String[] {"PLAIN"}, new PlainCallbackHandler(username,
                        password));
            client = new MemcachedClient(
                    new ConnectionFactoryBuilder().setProtocol(Protocol.BINARY)
                            .setAuthDescriptor(authDescriptor)
                            .build(),
                    AddrUtil.getAddresses(connectionaddress));
            String key = "memcached";//Stores data with the key being memcached in Memcached.
            String value = "Hello World";//The value is Hello World.
            int expireTime = 5; //Specifies the expiration time, measured in seconds. The countdown
starts from the moment data is written. After the expireTime elapses, the data expires and can no
longer be read.
            doExcute(client, key, value, expireTime);//Executes the operation.
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
    /**
     *Method of writing data to Memcached
     */
    private static void doExcute(MemcachedClient client, String key, String value, int expireTime)
    {
        try
        {
            OperationFuture<Boolean> future = client.set(key, expireTime, value);
            future.get();//spymemcached set () is asynchronous. future.get () waits until the cache.set
() operation is completed, or does not need to wait. You can select based on actual requirements.
            System.out.println("The Set operation succeeded.");
            System.out.println("Get operation:" + client.get(key));
            Thread.sleep(6000);//Waits for 6000 ms, that is, 6s. Then the data expires and can no longer
be read.
            System.out.println("Perform the Get operation 6s later:" + client.get(key));

        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        catch (ExecutionException e)
        {
            e.printStackTrace();
        }
        if (client != null)
        {
            client.shutdown();
        }
    }
}
```

- Example code for the password-free mode

  Change *ip:port* to the IP address and port number obtained in **Step 4**.

```
//Connect to the password-free Memcached code using Java.
import java.io.IOException;
import java.util.concurrent.ExecutionException;

import net.spy.memcached.AddrUtil;
import net.spy.memcached.BinaryConnectionFactory;
import net.spy.memcached.MemcachedClient;
import net.spy.memcached.internal.OperationFuture;

public class ConnectMemcached
{
    public static void main(String[] args)
    {
        final String connectionaddress = "ip:port";
        MemcachedClient client = null;
        try
        {
            client = new MemcachedClient(new BinaryConnectionFactory(),
AddrUtil.getAddresses(connectionaddress));
            String key = "memcached";//Stores data with the key being memcached in Memcached.
            String value = "Hello World";//The value is Hello World.
            int expireTime = 5; //Specifies the expiration time, measured in seconds. The countdown
starts from the moment data is written. After the expireTime elapses, the data expires and can no
longer be read.
            doExcute(client, key, value, expireTime);//Executes the operation.
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }

    /**
     *Method of writing data to Memcached
```

```
  */
    private static void doExcute(MemcachedClient client, String key, String value, int expireTime)
    {
        try
        {
            OperationFuture<Boolean> future = client.set(key, expireTime, value);
            future.get();//spymemcached set () is asynchronous. future.get () waits until the cache.set
() operation is completed, or does not need to wait. You can select based on actual requirements.
            System.out.println("The Set operation succeeded.");
            System.out.println("Get operation:" + client.get(key));
            Thread.sleep(6000);//Waits for 6000 ms, that is, 6s. Then the data expires and can no longer
be read.
            System.out.println("Perform the Get operation 6s later:" + client.get(key));

        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        catch (ExecutionException e)
        {
            e.printStackTrace();
        }
        if (client != null)
        {
            client.shutdown();
        }
    }
}
```

Step 9 Run the **main** method. The following result is displayed in the **Console** window of Eclipse:

```
The Set operation succeeded.
Get operation: Hello World
Perform the Get operation 6s later: null
```

----End

4.2.5.3 Connecting to Memcached on the Python-binary-memcached (Python)

Access a DCS Memcached instance using Python on an ECS in the same VPC.

Prerequisites

- The DCS Memcached instance you want to access is in the **Running** state.
- Log in to the ECS. For details on how to create ECSs, see the *Elastic Cloud Server User Guide*.

  NOTE

  An ECS can communicate with a DCS instance that belongs to the same VPC and is configured with the same security group.

  - If the ECS and DCS instance are not in the same VPC, you can establish a VPC peering connection to achieve network connectivity between the ECS and DCS instance. For details, see **Does DCS Support Cross-VPC Access?**
  - If different security groups have been configured for the ECS and DCS instance, you can set security group rules to achieve network connectivity between the ECS and DCS instance. For details, see **Security Group Configurations**.
- Python has been installed on the ECS. The recommended version is 2.7.6 or later.

- You have obtained the python-binary-memcached-x.y.z.zip dependency package.

  NOTE

  x.y.z indicates the version of the dependency package. The latest version is recommended.

Connecting to Memcached on the Python-binary-memcached

Step 1 Log in to the DCS console.

Step 2 Click in the upper left corner of the management console and select the region where your instance is located.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 On the **Cache Manager** page, click the name of the DCS Memcached instance you want to access. Obtain the IP address and port number of the instance.

Step 5 Upload the obtained dependency package (for example, the **python-binary-memcached-*x.y.z*.zip** package) to the created ECS.

Step 6 Log in to the ECS.

Step 7 Run the following commands to install the dependency package:

unzip -xzvf python-binary-memcached-x.y.z.zip

cd python-binary-memcached-x.y.z

python setup.py install

NOTE

If an error is reported during the installation, use the **apt** or **yum** installation method. For example, to install the dependency package by using the **apt** method, run the following commands:

apt install python-pip;

pip install python-binary-memcached;

Step 8 Create a Python file named **dcs_test.py**, copy the following Python code to the file, and modify the code.

- Example code for the password mode

  Change *ip:port* to the IP address and port number obtained in **Step 4**. Set *userName* and *password* respectively to the username and password of the Memcached instance.

```
import bmemcached
client = bmemcached.Client(('ip:port'), 'userName', 'password')
print "set('key', 'hello world!')"
print client.set('key', 'hello world!')
print "get('key')"
print client.get('key')
```

- Example code for the password-free mode

  Change ip:port to the IP address and port number obtained in **Step 4**.

```
import bmemcached
client = bmemcached.Client('ip:port')
print "set('key', 'hello world!')"
print client.set('key', 'hello world!')
print "get('key')"
print client.get('key')
```

Step 9 Run the **dcs_test.py** file. The following result is displayed.

```
# python test.py
set('key', 'hello world!')
True
get('key')
hello world!
```

----End

4.2.5.4 Connecting to Memcached on the Libmemcached (C++)

Access a DCS Memcached instance using a C++ client on an ECS in the same VPC.

Prerequisites

- The DCS Memcached instance you want to access is in the **Running** state.
- Log in to the ECS. For details on how to create ECSs, see the *Elastic Cloud Server User Guide*.

  NOTE

  An ECS can communicate with a DCS instance that belongs to the same VPC and is configured with the same security group.

  - If the ECS and DCS instance are not in the same VPC, you can establish a VPC peering connection to achieve network connectivity between the ECS and DCS instance. For details, see **Does DCS Support Cross-VPC Access?**
  - If different security groups have been configured for the ECS and DCS instance, you can set security group rules to achieve network connectivity between the ECS and DCS instance. For details, see **Security Group Configurations**.
- GCC has been installed on the ECS. The recommended version is 4.8.4 or later.
- You have obtained the **libmemcached-x.y.z.tar.gz** dependency package.

  NOTE

  x.y.z indicates the version of the dependency package. The latest version is recommended.

Connecting to Memcached on the Libmemcached (C++)

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner of the management console and select the region where your instance is located.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 On the **Cache Manager** page, click the name of the DCS Memcached instance you want to access. Obtain the IP address and port number of the instance.

Step 5 Upload the obtained **libmemcached-***x.y.z***.tar.gz** dependency package to the created ECS.

Step 6 Log in to the ECS.

Step 7 Install related SASL dependency packages.

For OSs of Debian series: **apt install libsasl2-dev cloog.ppl**

For OSs of Red Hat series: **yum install cyrus-sasl***

Step 8 Run the following commands to install the dependency package:

tar -xzvf libmemcached-x.y.z.tar.gz

cd libmemcached-x.y.z

./configure --enable-sasl

make

make install

Step 9 Create a file named **build.sh** and copy the following code to the file.

```
g++ -o dcs_sample dcs_sample.cpp -lmemcached -std=c++0x -lpthread -lsasl2
```

NOTE

If the **libmemcached.so.11** file cannot be found during compilation, run the **find** command to find the file and copy the file to the **/usr/lib** directory.

Step 10 Create a file named **dcs_sample.cpp**, copy the following C++ code to the file, and modify the code.

- Example code for the password mode

  Change *ip* and *port* to the IP address and port number obtained in **Step 4**. Set *userName* and *password* respectively to the username and password of the Memcached instance.

```
#include <iostream>
#include <string>
#include <libmemcached/memcached.h>
using namespace std;

#define IP  "ip"
#define PORT  "port"
#define USERNAME "userName"
#define PASSWORD "password"
memcached_return result;

memcached_st * init()
{
    memcached_st *memcached = NULL;
    memcached_server_st *cache;
    memcached = memcached_create(NULL);
    cache = memcached_server_list_append(NULL, IP, PORT, &result);

    sasl_client_init(NULL);
         memcached_set_sasl_auth_data(memcached, USERNAME, PASSWORD);
         memcached_behavior_set(memcached,MEMCACHED_BEHAVIOR_BINARY_PROTOCOL,1);
         memcached_server_push(memcached,cache);
         memcached_server_list_free(cache);
            return memcached;
}

int main(int argc, char *argv[])
{
         memcached_st *memcached=init();
    string key = "memcached";
    string value = "hello world!";
    size_t value_length = value.length();
    int expire_time = 0;
    uint32_t flag = 0;

    result =
memcached_set(memcached,key.c_str(),key.length(),value.c_str(),value.length(),expire_time,flag);
    if (result != MEMCACHED_SUCCESS){
      cout <<"set data failed: " << result << endl;
      return -1;
    }
    cout << "set succeed, key: " << key << ", value: " << value << endl;
    cout << "get key:" << key << endl;
    char* result = memcached_get(memcached,key.c_str(),key.length(),&value_length,&flag,&result);
    cout << "value:" << result << endl;

    memcached_free(memcached);
    return 0;
}
```

- Example code for the password-free mode

  Change ip and *port* to the IP address and port number obtained in **Step 4**.

```
#include <iostream>
#include <string>
#include <libmemcached/memcached.h>
using namespace std;

#define IP  "ip"
#define PORT  port
memcached_return result;

memcached_st * init()
{
    memcached_st *memcached = NULL;
    memcached_server_st *cache;
    memcached = memcached_create(NULL);
    cache = memcached_server_list_append(NULL, IP, PORT, &result);
         memcached_server_push(memcached,cache);
    memcached_server_list_free(cache);
         return memcached;
}

int main(int argc, char *argv[])
{
         memcached_st *memcached=init();
    string key = "memcached";
    string value = "hello world!";
    size_t value_length = value.length();
    int expire_time = 0;
    uint32_t flag = 0;

    result =
memcached_set(memcached,key.c_str(),key.length(),value.c_str(),value.length(),expire_time,flag);
    if (result != MEMCACHED_SUCCESS){
      cout <<"set data failed: " << result << endl;
      return -1;
    }
    cout << "set succeed, key: " << key << " ,value: " << value << endl;
    cout << "get key:" << key << endl;
    char* result = memcached_get(memcached,key.c_str(),key.length(),&value_length,&flag,&result);
    cout << "value:" << result << endl;

    memcached_free(memcached);
    return 0;
}
```

Step 11 Run the following commands to compile the source code:

chmod 700 build.sh

./build.sh

The **dcs_sample** binary file is generated.

Step 12 Run the following command to access the chosen DCS Memcached instance:

./dcs_sample

```
set succeed, key: memcached ,value: hello world!
get key:memcached
value:hello world!
```

----End

4.2.5.5 Connecting to Memcached on the Libmemcached (PHP)

Access a DCS Memcached instance in PHP on an ECS in the same VPC.

Prerequisites

- The DCS Memcached instance you want to access is in the **Running** state.
- Log in to the ECS. For details on how to create ECSs, see the *Elastic Cloud Server User Guide*.

> NOTE
>
> An ECS can communicate with a DCS instance that belongs to the same VPC and is configured with the same security group.
>
> - If the ECS and DCS instance are not in the same VPC, you can establish a VPC peering connection to achieve network connectivity between the ECS and DCS instance. For details, see **Does DCS Support Cross-VPC Access?**
> - If different security groups have been configured for the ECS and DCS instance, you can set security group rules to achieve network connectivity between the ECS and DCS instance. For details, see **Security Group Configurations**.

OSs of Red Hat Series

The following uses CentOS 7.0 as an example to describe how to install a PHP client and use it to access a DCS Memcached instance. The procedure is also applicable to a PHP client running the Red Hat or Fedora OS.

Step 1 Install GCC-C++ and Make compilation components.

yum install gcc-c++ make

Step 2 Install related SASL packages.

yum install cyrus-sasl*

Step 3 Install the libMemcached library.

Installing the libMemcached library requires SASL authentication parameters. Therefore, you cannot install the library by running the **yum** command.

wget https://launchpad.net/libmemcached/1.0/1.0.18/+download/libmemcached-1.0.18.tar.gz

tar -xvf libmemcached-1.0.18.tar.gz

cd libmemcached-1.0.18

./configure --prefix=/usr/local/libmemcached --enable-sasl

make && make install

NOTE

Before installing the libMemcached library, install GCC-C++ and SASL components. Otherwise, an error will be reported during compilation. After you resolve the error, run the **make clean** command and then run the **make** command again.

Step 4 Install the PHP environment.

yum install php-devel php-common php-cli

NOTICE

PHP 7.x does not support SASL authentication. Use PHP 5.6. If the yum php version is not 5.6, download one from the Internet.

Step 5 Install the Memcached client.

Note that you must add a parameter used to enable SASL when running the **configure** command.

wget http://pecl.php.net/get/memcached-2.1.0.tgz

tar zxvf memcached-2.1.0.tgz

cd memcached-2.1.0

phpize

./configure --with-libmemcached-dir=/usr/local/libmemcached --enable-memcached-sasl

make && make install

Step 6 Modify the **php.ini** file.

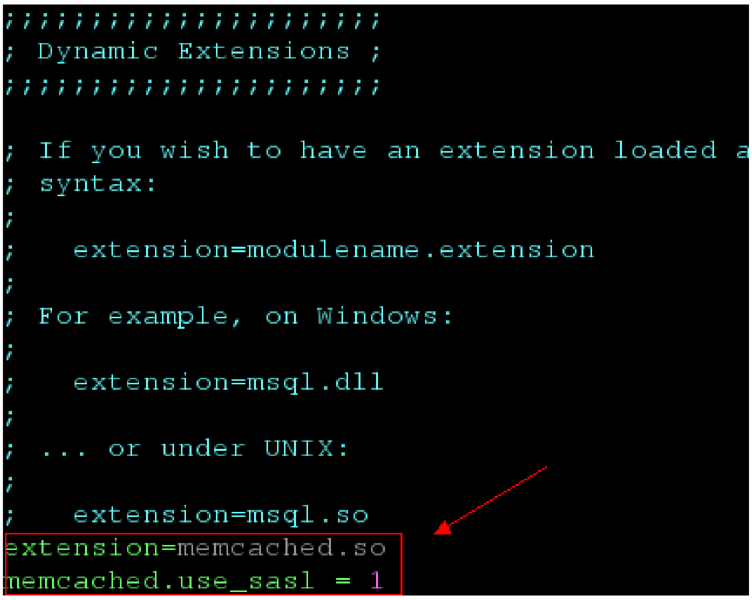
Run the **find** or **locate** command to find the **php.ini** file.

find / -name php.ini

Add the following two lines to the **php.ini** file:

```
extension=memcached.so
memcached.use_sasl = 1
```

Figure 4-7 Modifying the **php.ini** file



```
;;;;;;;;;;;;;;;;;;;;;;
; Dynamic Extensions ;
;;;;;;;;;;;;;;;;;;;;;;

; If you wish to have an extension loaded a
; syntax:
;
;   extension=modulename.extension
;
; For example, on Windows:
;
;   extension=msql.dll
;
; ... or under UNIX:
;
;   extension=msql.so
extension=memcached.so
memcached.use_sasl = 1
```

Step 7 Access a DCS Memcached instance.

Create a **memcached.php** file and add the following content to the file:

```
<?php
   $connect = new Memcached; //Declares a Memcached connection.
   $connect->setOption(Memcached::OPT_COMPRESSION, false); //Disables compression.
   $connect->setOption(Memcached::OPT_BINARY_PROTOCOL, true); //Uses the binary protocol.
   $connect->setOption(Memcached::OPT_TCP_NODELAY, true); //Disables the TCP network delay policy.
   $connect->addServer('{memcached_instance_ip}', 11211); //Specifies the instance IP address and port
number.
   $connect->setSaslAuthData('{username}', '{password}'); //If password-free access is enabled for the
instance, delete or comment out this line.
   $connect->set("DCS", "Come on!");
   echo 'DCS: ',$connect->get("DCS");
   echo "\n";
   $connect->quit();
?>
```

Save and run the **memcached.php** file. The following result is displayed.

```
[root@testphpmemcached ~]# php memcached.php
   DCS: Come on!
[root@testphpmemcached ~]#
```

----End

OSs of Debian Series

The following uses the Ubuntu OS as an example to describe how to install a PHP client and use it to access a DCS Memcached instance.

Step 1 Install GCC and Make compilation components.

apt install gcc make

Step 2 Install the PHP environment.

PHP 5.x is recommended for better compatibility with SASL authentication.

Run the following commands to add the image source of PHP of an earlier version, and then install the **php.5.6** and **php.5.6-dev** packages:

apt-get install -y language-pack-en-base;

LC_ALL=en_US.UTF-8;

add-apt-repository ppa:ondrej/php;

apt-get update;

apt-get install php5.6 php5.6-dev;

After the installation is complete, run the **php -version** command to check the PHP version. If the following result is displayed, the PHP version is 5.6, indicating that PHP 5.6 is successfully installed.

```
root@dcs-nodelete:/etc/apt# php -version
PHP 5.6.36-1+ubuntu16.04.1+deb.sury.org+1 (cli)
Copyright (c) 1997-2016 The PHP Group
```

NOTE

To uninstall PHP, run the following commands:

apt install aptitude -y

aptitude purge `dpkg -l | grep php| awk '{print $2}' |tr "\n" " "`

Step 3 Install the SASL component.

apt install libsasl2-dev cloog.ppl

Step 4 Install the libMemcached library.

wget https://launchpad.net/libmemcached/1.0/1.0.18/+download/libmemcached-1.0.18.tar.gz

tar -xvf libmemcached-1.0.18.tar.gz

cd libmemcached-1.0.18

./configure --prefix=/usr/local/libmemcached

make && make install

NOTE

Before installing the libMemcached library, install GCC-C++ and SASL components. Otherwise, an error will be reported during compilation. After you resolve the error, run the **make clean** command and then run the **make** command again.

Step 5 Install the Memcached client.

Install the zlib component.

apt install zlib1g.dev

Note that you must add a parameter used to enable SASL when running the **configure** command.

wget http://pecl.php.net/get/memcached-2.2.0.tgz;

tar zxvf memcached-2.2.0.tgz;

cd memcached-2.2.0;

phpize5.6;

./configure --with-libmemcached-dir=/usr/local/libmemcached --enable-memcached-sasl;

make && make install;

Step 6 Modify the **pdo.ini** file.

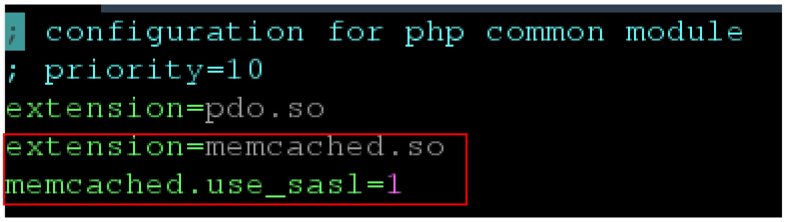
Run the following command to find the **pdo.ini** file:

find / -name pdo.ini

By default, the **pdo.ini** file is stored in the **/etc/php/5.6/mods-available** directory. Add the following two lines to the **php.ini** file:

```
extension=memcached.so
memcached.use_sasl = 1
```

Figure 4-8 Modifying the **pdo.ini** file



```
; configuration for php common module
; priority=10
extension=pdo.so
extension=memcached.so
memcached.use_sasl=1
```

Step 7 Access a DCS Memcached instance.

Create a **memcached.php** file and add the following content to the file:

```
<?php
    $connect = new Memcached; //Declares a Memcached connection.
    $connect->setOption(Memcached::OPT_COMPRESSION, false); //Disables compression.
    $connect->setOption(Memcached::OPT_BINARY_PROTOCOL, true); //Uses the binary protocol.
    $connect->setOption(Memcached::OPT_TCP_NODELAY, true); //Disables the TCP network delay policy.
    $connect->addServer('{memcached_instance_ip}', 11211); //Specifies the instance IP address and port
number.
    $connect->setSaslAuthData('{username}', '{password}'); //If password-free access is enabled for the
instance, delete or comment out this line.
    $connect->set("DCS", "Come on!");
    echo 'DCS: ',$connect->get("DCS");
    echo "\n";
    $connect->quit();
?>
```

Save and run the **memcached.php** file. The following result is displayed.

```
[root@dcs-nodelete ~]# php memcached.php
    DCS: Come on!
[root@dcs-nodelete ~]#
```

----End

4.3 Viewing Details of a DCS Instance

On the DCS console, you can view DCS instance details.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Search for DCS instances using any of the following methods:

- Search by keyword.

  If you do not select an attribute and enter a keyword in the search box, the system searches for instances by instance name by default.

- Select attributes and enter their keywords to search.

  Click the search box, select an instance attribute, and enter a keyword to search. You can select multiple attributes at a time.

  For example, choose **Status** > **Running**, **Type** > **Master/Standby**, and **Cache Engine** > **Redis 5.0**.

For more information on how to search, click the question mark to the right of the search box.

Step 5 On the DCS instance list, click the name of a DCS instance to display more details about it. The following table describes the parameters.

Table 4-18 Parameters on the Basic Information page of a DCS instance

Section	Parameter	Description
Instance Details	Name	Name of the DCS instance. To modify the instance name, click ✎.
	Status	State of the chosen instance.
	ID	ID of the chosen instance.
	Cache Engine	Cache engine used by the DCS instance, which can be Redis or Memcached. If the cache engine is Redis, it is followed by the version number, for example, Redis 5.0.
	Instance Type	Type of the selected instance. Currently, supported types include single-node, master/standby, Proxy Cluster, and Redis Cluster.
	Cache Size	Specification of the chosen instance.
	Used/ Available Memory (MB)	The used memory space and maximum available memory space of the chosen instance. The used memory space includes: ● Size of data stored on the DCS instance ● Size of Redis-server buffers (including client buffer and repl-backlog) and internal data structures
	CPU	CPU of the DCS instance.

Section	Parameter	Description
	Enterprise Project	Enterprise project to which the instance belongs. Click ✎ to modify the enterprise project of the instance.
	Description	Description of the DCS instance. To modify the description, click ✎.
Connection	Password Protected	Currently, password-protected access and password-free access are supported.
	IP Address	IP address and port number of the chosen instance.
Network	AZ	Availability zone in which the cache node running the selected DCS instance resides.
	VPC	VPC in which the chosen instance resides.
	Subnet	Subnet in which the chosen instance resides.
	Security Group	Security group that controls access to the chosen instance. Only Redis 3.0 supports access control with security groups. To modify the security group, click ✎. DCS Redis 4.0 and later are based on VPC Endpoint and do not support security groups. You can click **configure the whitelist** to configure the whitelist.
Billing	Billing Mode	Pay-per-use
	Created	Time at which the chosen instance started to be created.
	Run	Time at which the instance was created.
Instance Topology	-	Hover the mouse pointer over an instance to view its metrics, or click the icon of an instance to view its historical metrics. Topologies are supported only for master/standby and cluster instances.

----End

5 Operating DCS Instances

5.1 Modifying DCS Instance Specifications

On the DCS console, you can scale a DCS Redis or Memcached instance to a larger or smaller capacity.

NOTE

- **Modify instance specifications during off-peak hours.** If the modification failed in peak hours (for example, when memory or CPU usage is over 90% or write traffic surges), try again during off-peak hours.
- You can only change the instance type of single-node or master/standby DCS Redis 3.0 instances.
- If your DCS instances are too old to support scaling, contact technical support to upgrade the instances.
- Services may be interrupted for seconds during the modification. Therefore, services connected to Redis must support reconnection.
- Modifying instance specifications does not affect the connection address, password, data, security group, and whitelist configurations of the instance.

Change of the Instance Type

- **Supported instance type changes:**
  - From single-node to master/standby: Supported by Redis 3.0 and Memcached, and not by Redis 4.0/5.0/6.0.
  - From master/standby to Proxy Cluster: Supported by Redis 3.0, and not by Redis 4.0/5.0/6.0.

    If the data of a master/standby DCS Redis 3.0 instance is stored in multiple databases, or in non-DB0 databases, the instance cannot be changed to the Proxy Cluster type. A master/standby instance can be changed to the Proxy Cluster type only if its data is stored only on DB0.
  - From cluster types to other types: Not supported.
- **Impact of instance type changes:**
  - From single-node to master/standby for a DCS Redis 3.0 instance:

    The instance cannot be connected for several seconds and remains read-only for about 1 minute.

- From master/standby to Proxy Cluster for a DCS Redis 3.0 instance:

  The instance cannot be connected and remains read-only for 5 to 30 minutes.

Scaling

- **The following table lists scaling options supported by different DCS instances.**

Table 5-1 Scaling options supported by different DCS instances

Cache Engine	Single-Node	Master/ Standby	Redis Cluster	Proxy Cluster
Redis 3.0	Scaling up/ down	Scaling up/ down	N/A	Scaling out
Redis 4.0	Scaling up/ down	Scaling up/ down and replica quantity change	Scaling up/ out, down/in, and replica quantity change	N/A
Redis 5.0	Scaling up/ down	Scaling up/ down and replica quantity change	Scaling up/ out, down/in, and replica quantity change	N/A
Redis 6.0	Scaling up/ down	Scaling up/ down	N/A	N/A
Memcached	Scaling up/ down	Scaling up/ down	N/A	N/A

NOTE

If the reserved memory of a DCS Redis 3.0 or Memcached instance is insufficient, the scaling may fail when the memory is used up.

Change the replica quantity and capacity separately.

- **Impact of scaling:**
  - Single-node and master/standby
    - A DCS Redis 4.0/5.0/6.0 instance will be disconnected for several seconds and remain read-only for about 1 minute.
    - A DCS Redis 3.0 instance will be disconnected and remain read-only for 5 to 30 minutes.
    - For scaling up, only the memory of the instance is expanded. The CPU processing capability is not improved.

        - Data of single-node instances may be lost because they do not support data persistence. After scaling, check whether the data is complete and import data if required.
        - Backup records of master/standby instances cannot be used after scaling down.
    - Cluster
        - If the shard quantity is not decreased, the instance can always be connected, but the CPU usage will increase, compromising performance by up to 20%, and the latency will increase during data migration.
        - During scaling up, new Redis Server nodes are added, and data is automatically balanced to the new nodes.
        - Nodes will be deleted if the shard quantity decreases. To prevent disconnection, ensure that the deleted nodes are not directly referenced in your application.
        - Ensure that the used memory of each node is less than 70% of the maximum memory per node of the new flavor. Otherwise, you cannot perform the scale-in.
        - If the memory becomes full during scaling due to a large amount of data being written, scaling will fail. Modify specifications during off-peak hours.
        - Scaling involves data migration. The latency for accessing the key being migrated increases. For a Redis Cluster instance, ensure that the client can properly process the **MOVED** and **ASK** commands. Otherwise, requests will fail.
        - Before scaling, perform cache analysis to ensure that no big keys (≥ 512 MB) exist in the instance. Otherwise, scaling may fail.
        - Backup records created before scaling cannot be restored.
- **Notes on changing the number of replicas of a DCS Redis instance:**

  Deleting replicas interrupts connections. If your application cannot reconnect to Redis or handle exceptions, you need to restart the application after scaling.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Choose **More** > **Modify Specifications** in the row containing the DCS instance.

Step 5 On the **Modify Specifications** page, select the desired specification.

NOTE

For a master/standby DCS Redis 4.0/5.0 instance or a Redis Cluster instance, you can choose to change by specification or replica quantity.

Step 6 Click **Submit** to start modifying the DCS instance.

You can go to **Background Tasks** page to view the modification status. For more information, see **Viewing Background Tasks**.

Specification modification of a single-node or master/standby DCS instance takes about 5 to 30 minutes to complete, while that of a cluster DCS instance takes a longer time. After an instance is successfully modified, it changes to the **Running** state.

NOTE

- If the specification modification of a single-node DCS instance fails, the instance is temporarily unavailable for use. The specification remains unchanged. Some management operations (such as parameter configuration and specification modification) are temporarily not supported. After the specification modification is completed in the backend, the instance changes to the new specification and becomes available for use again.
- If the specification modification of a master/standby or cluster DCS instance fails, the instance is still available for use with its original specifications. Some management operations (such as parameter configuration, backup, restoration, and specification modification) are temporarily not supported. Remember not to read or write more data than allowed by the original specifications; otherwise, data loss may occur.
- After the specification modification is successful, the new specification of the instance takes effect.

----End

5.2 Restarting DCS Instances

On the DCS console, you can restart one or multiple DCS instances at a time.

NOTICE

- After a single-node DCS instance is restarted, data will be deleted from the instance.
- While a DCS instance is restarting, it cannot be read from or written to.
- An attempt to restart a DCS instance while it is being backed up may result in a failure.
- Restarting a DCS instance will disconnect the original client. You are advised to configure automatic reconnection in your application.

Prerequisites

The DCS instances you want to restart are in the **Running** or **Faulty** state.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 On the **Cache Manager** page, select one or more DCS instances you want to restart.

Step 5 Click **Restart** above the DCS instance list.

Step 6 In the displayed dialog box, click **Yes**.

It takes 10 seconds to 30 minutes to restart DCS instances. After DCS instances are restarted, their status changes to **Running**.

NOTE

- To restart a single instance, you can also click **Restart** in the same row as the instance.
- The time required for restarting a DCS instance depends on the cache size of the instance.

----End

5.3 Deleting DCS Instances

On the DCS console, you can delete one or multiple DCS instances at a time. You can also delete all instance creation tasks that have failed to run.

NOTICE

- After a DCS instance is deleted, the instance data will also be deleted without backup. In addition, any backup data of the instance will be deleted. Therefore, download the backup files of the instance for permanent storage before deleting the instance.
- If the instance is in cluster mode, all cluster nodes will be deleted.

Prerequisites

- The DCS instances you want to delete have been created.
- The DCS instances you want to delete are in the **Running** or **Faulty** state.

Procedure

Deleting DCS Instances

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 On the **Cache Manager** page, select one or more DCS instances you want to delete.

DCS instances in the **Creating**, **Restarting**, **Upgrading**, **Resizing**, **Clearing data**, **Backing up**, or **Restoring** state cannot be deleted.

Step 5 Choose **More** > **Delete** above the instance list.

Step 6 Enter **DELETE** as prompted and click **Yes** to delete the instance.

It takes 1 to 30 minutes to delete DCS instances.

> NOTE
>
> To delete a single instance, choose **Operation** > **More** > **Delete** in the same row as the instance.

----End

Deleting Instance Creation Tasks That Have Failed to Run

Step 1 Log in to the DCS console.

Step 2 Click in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 If there are DCS instances that have failed to be created, **Instance Creation Failures** is displayed above the instance list.

Step 5 Click the icon or the number of failed tasks next to **Instance Creation Failures**.

The **Instance Creation Failures** dialog box is displayed.

Step 6 Choose failed instance creation tasks to delete.

- To delete all failed tasks, click **Delete All** above the task list.
- To delete a single failed task, click **Delete** in the same row as the task.

----End

5.4 Performing a Master/Standby Switchover for a DCS Instance

On the DCS console, you can manually switch the master and standby nodes of a DCS instance. This operation is used for special purposes, for example, releasing all service connections or terminating ongoing service operations.

Only master/standby instances support a master/standby node switchover.

For DCS Redis 4.0 and later cluster instances, if you need to manually switch master/standby nodes on a shard, go to the **Shards and Replicas** page of the instance. For details, see **Managing Shards and Replicas**.

NOTICE

- Services may be interrupted for up to 10 seconds during the switchover. Before performing a switchover, ensure that your application supports reconnection.
- During a master/standby node switchover, a large amount of resources will be consumed for data synchronization between the master and standby nodes. You are advised to perform this operation during off-peak hours.
- Data of the maser and standby nodes is synchronized asynchronously. Therefore, a small amount of data that is being operated on during the switchover may be lost.

Prerequisites

The DCS instance for which you want to perform a master/standby node switchover is in the **Running** state.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 In the **Operation** column of the instance, choose **More** > **Master/Standby Switchover**, then click **OK**.

----End

5.5 Clearing DCS Instance Data

You can clear data of DCS Redis 4.0 and later instances on the console.

Clearing instance data cannot be undone and cleared data cannot be recovered. Exercise caution when performing this operation.

Prerequisites

The instance is in the **Running** state.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Select one or more DCS instances.

Step 5 Click **Clear data** above the instance list.

Step 6 In the displayed dialog box, click **Yes**.

----End

5.6 Exporting DCS Instance List

On the DCS console, you can export DCS instance information in full to an Excel file.

Procedure

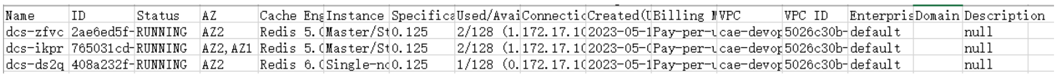
Step 1 Log in to the DCS console.

Step 2 Click in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click **Import** above the instance list to export the DCS instance list.

Figure 5-1 DCS instance list



Name	ID	Status	AZ	Cache Eng	Instance	Specifica	Used/Avai	Connectic	Created(U	Billing M	VPC	VPC ID	Enterpris	Domain	Description
dcs-zfvc	2ae6ed5f-	RUNNING	AZ2	Redis 5.(	Master/St	0.125	2/128 (1.	172.17.1(	2023-05-1	Pay-per-u	cae-devop	5026c30b-	default		null
dcs-ikpr	765031cd-	RUNNING	AZ2,AZ1	Redis 5.(	Master/St	0.125	2/128 (1.	172.17.1(	2023-05-(	Pay-per-u	cae-devop	5026c30b-	default		null
dcs-ds2q	408a232f-	RUNNING	AZ2	Redis 6.(	Single-nc	0.125	1/128 (0.	172.17.1(	2023-05-1	Pay-per-u	cae-devop	5026c30b-	default		null

----End

5.7 Command Renaming

After creating a DCS Redis 4.0 or later instance, you can rename the following critical commands: Currently, you can only rename the **COMMAND**, **KEYS**, **FLUSHDB**, **FLUSHALL**, **HGETALL**, **SCAN**, **HSCAN**, **SSCAN**, and **ZSCAN** commands.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 In the **Operation** column of an instance, choose **More** > **Command Renaming**.

Step 5 Select a command, enter a new name, and click **OK**.

In the **Command Renaming** dialog box, click **Add Command** to rename multiple commands at the same time.

NOTE

- Remember the new command names because they will not be displayed on the console for security purposes.
- The system will restart the instance after you rename commands. The new commands take effect after the restart.
- To use the original name of a command, rename the command again.
- The new name can contain 4 to 64 characters including letters, digits, underscores (_), and hyphens (-), and must start with a letter.

----End

6 Managing DCS Instances

6.1 Configuration Notice

- In most cases, different DCS instance management operations cannot proceed concurrently. If you initiate a new management operation while the current operation is in progress, the DCS console prompts you to initiate the new operation again after the current operation is complete. DCS instance management operations include:
  - Creating a DCS instance
  - Configuring parameters
  - Restarting a DCS instance
  - Changing the instance password
  - Resetting the instance password
  - Scaling, backing up, or restoring an instance
- You can restart a DCS instance while it is being backed up, but the backup task will be forcibly interrupted and is likely to result in a backup failure.

NOTICE

In the event that a cache node of a DCS instance is faulty:

- The instance remains in the **Running** state and you can continue to read from and write to the instance. This is achieved thanks to the high availability of DCS.
- Cache nodes can recover from internal faults automatically. Manual fault recovery is also supported.
- Certain operations (such as parameter configuration, password change or resetting, backup, restoration, and specification modification) in the management zone are not supported during fault recovery. You can contact technical support or perform these operations after the cache nodes recover from faults.

6.2 Modifying Configuration Parameters

You can modify the configuration parameters of your DCS instance to optimize DCS performance based on your requirements.

For example, if you do not need data persistence, set **appendonly** to **no**.

NOTE

After the instance configuration parameters are modified, the modification takes effect immediately without the need to manually restart the instance. For a cluster instance, the modification takes effect on all shards.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 On the **Cache Manager** page, click the name of the DCS instance you want to configure.

Step 5 Choose **Instance Configuration** > **Parameters**.

Step 6 On the **Parameters** page, click **Modify**.

Step 7 Modify parameters based on your requirements.

Table 6-1 and Table 6-2 describe the parameters. In most cases, retain the default values.

Table 6-1 DCS Redis instance configuration parameters

Parameter	Description	Value Range	Default Value
active-expire-num	Number of expired keys that can be deleted in regular scans. Redis 3.0 instances do not have this parameter.	1–1000	20

Parameter	Description	Value Range	Default Value
timeout	The maximum amount of time (in seconds) a connection between a client and the DCS instance can be allowed to remain idle before the connection is terminated. A setting of **0** means that this function is disabled. Proxy Cluster instances do not have this parameter.	0–7200 seconds	0
appendfsync	Controls how often fsync() transfers cached data to the disk. Note that some OSs will perform a complete data transfer but some others only make a "best-effort" attempt. There are three settings: no: fsync() is never called. The OS will flush data when it is ready. This mode offers the highest performance. always: fsync() is called after every write to the AOF. This mode is very slow, but also very safe. everysec: fsync() is called once per second. This mode provides a compromise between safety and performance. Single-node instances do not have this parameter.	● no ● always ● everysec	no

Parameter	Description	Value Range	Default Value
appendonly	Indicates whether to log each modification of the instance (that is, data persistence). By default, data is written to disks asynchronously in Redis. If this function is disabled, recently-generated data might be lost in the event of a power failure. Options: yes: enabled no: disabled Single-node instances do not have this parameter.	● yes ● no	yes
client-output-buffer-limit-slave-soft-seconds	Number of seconds that the output buffer remains above **client-output-buffer-slave-soft-limit** before the client is disconnected. Single-node instances do not have this parameter.	0–60	60
client-output-buffer-slave-hard-limit	Hard limit (in bytes) on the output buffer of replica clients. Once the output buffer exceeds the hard limit, the client is immediately disconnected. Single-node instances do not have this parameter.	Depends on the instance type and specifications.	Depends on the instance type and specificat ions.

Parameter	Description	Value Range	Default Value
client-output-buffer-slave-soft-limit	Soft limit (in bytes) on the output buffer of replica clients. Once the output buffer exceeds the soft limit and continuously remains above the limit for the time specified by the **client-output-buffer-limit-slave-soft-seconds** parameter, the client is disconnected. Single-node instances do not have this parameter.	Depends on the instance type and specifications.	Depends on the instance type and specificat ions.

Parameter	Description	Value Range	Default Value
maxmemory-policy	The deletion policy to apply when the maxmemory limit is reached. Options: **volatile-lru**: Evict keys by trying to remove the less recently used (LRU) keys first, but only among keys that have an expire set. **(Recommended)** **allkeys-lru**: Evict keys by trying to remove the LRU keys first. **volatile-random**: evict keys randomly, but only evict keys with an expire set. **allkeys-random**: Evict keys randomly. **volatile-ttl**: Evict keys with an expire set, and try to evict keys with a shorter time to live (TTL) first. **noeviction**: Do not delete any keys and only return errors when the memory limit was reached. **volatile-lfu**: Evict keys by trying to remove the less frequently used (LFU) keys first, but only among keys that have an expire set. **allkeys-lfu**: Evict keys by trying to remove the LFU keys first.	Depends on the instance version.	Depends on the instance version and type.
lua-time-limit	Maximum time allowed for executing a Lua script (in milliseconds).	100–5000	5,000

Parameter	Description	Value Range	Default Value
master-read-only	Sets the instance to be read-only. All write operations will fail. Proxy Cluster instances do not have this parameter.	● yes ● no	no
maxclients	The maximum number of clients allowed to be concurrently connected to a DCS instance. Proxy Cluster instances do not have this parameter.	Depends on the instance type and specifications.	Depends on the instance type and specifications.
proto-max-bulk-len	Maximum size of a single element request (in bytes).	1,048,576–536,870,912	536,870,912
repl-backlog-size	The replication backlog size (bytes). The backlog is a buffer that accumulates replica data when replicas are disconnected from the master. When a replica reconnects, a partial synchronization is performed to synchronize the data that was missed while replicas were disconnected. Single-node instances do not have this parameter.	16,384–1,073,741,824	1,048,576
repl-backlog-ttl	The amount of time, in seconds, before the backlog buffer is released, starting from the last a replica was disconnected. The value **0** indicates that the backlog is never released. Single-node instances do not have this parameter.	0–604,800	3,600

Parameter	Description	Value Range	Default Value
repl-timeout	Replication timeout (in seconds). Single-node instances do not have this parameter.	30–3,600	60
hash-max-ziplist-entries	Hashes are encoded using a memory efficient data structure when the number of entries in hashes is less than the value of this parameter.	1–10,000	512
hash-max-ziplist-value	Hashes are encoded using a memory efficient data structure when the biggest entry in hashes does not exceed the length threshold indicated by this parameter.	1–10,000	64
set-max-intset-entries	When a set is composed of just strings that happen to be integers in radix 10 in the range of 64 bit signed integers, sets are encoded using a memory efficient data structure.	1–10,000	512
zset-max-ziplist-entries	Sorted sets are encoded using a memory efficient data structure when the number of entries in sorted sets is less than the value of this parameter.	1–10,000	128
zset-max-ziplist-value	Sorted sets are encoded using a memory efficient data structure when the biggest entry in sorted sets does not exceed the length threshold indicated by this parameter.	1–10,000	64

Parameter	Description	Value Range	Default Value
latency-monitor-threshold	Threshold time in latency monitoring. Unit: millisecond. Set to **0**: Latency monitoring is disabled. Set to more than 0: All with at least this many milliseconds of latency will be logged. By running the **LATENCY** command, you can perform operations related to latency monitoring, such as obtaining statistical data, and configuring and enabling latency monitoring. Proxy Cluster instances do not have this parameter.	0–86,400,000 ms	0

Parameter	Description	Value Range	Default Value
notify-keyspace-events	Controls which keyspace events notifications are enabled for. If the value is an empty string, this function is disabled. A combination of different values can be used to enable notifications for multiple event types. Possible values: **K**: Keyspace events, published with the **__keyspace@__** prefix. **E**: Keyevent events, published with __keyevent@__ prefix **g**: Generic commands (non-type specific) such as DEL, EXPIRE, and RENAME **$**: String commands **l**: List commands **s**: Set commands **h**: Hash commands **z**: Sorted set commands **x**: Expired events (events generated every time a key expires) **e**: Evicted events (events generated when a key is evicted for maxmemory) **A**: an alias for "g$lshzxe" The parameter value must contain either **K** or **E**. **A** cannot be used together with any of the characters in "g$lshzxe". For example, the value **Kl** means that Redis will notify Pub/Sub clients about keyspace events and list commands. The value **AKE** means Redis will notify Pub/Sub clients about all events.	See the parameter description.	Ex

Parameter	Description	Value Range	Default Value
	Proxy Cluster instances do not have this parameter.		
slowlog-log-slower-than	Redis records queries that exceed a specified execution time. **slowlog-log-slower-than** is the maximum time allowed, in microseconds, for command execution. If this threshold is exceeded, Redis will record the query.	0–1,000,000	10,000
slowlog-max-len	The maximum allowed number of slow queries that can be logged. Slow query log consumes memory, but you can reclaim this memory by running the **SLOWLOG RESET** command.	0–1000	128

NOTE

1. For more information about the parameters described in **Table 6-1**, visit **https://redis.io/topics/memory-optimization**.
2. The **latency-monitor-threshold** parameter is usually used for fault location. After locating faults based on the latency information collected, change the value of **latency-monitor-threshold** to **0** to avoid unnecessary latency.
3. More about the **notify-keyspace-events** parameter:
   - The parameter setting must contain at least a **K** or **E**.
   - **A** is an alias for "g$lshzxe" and cannot be used together with any of the characters in "g$lshzxe".
   - For example, the value **Kl** means that Redis will notify Pub/Sub clients about keyspace events and list commands. The value **AKE** means Redis will notify Pub/Sub clients about all events.
4. Configurable parameters and their values vary depending on the instance type.

Table 6-2 DCS Memcached instance configuration parameters

Parameter	Description	Value Range	Default Value
timeout	The maximum amount of time (in seconds) a connection between a client and the DCS instance can be allowed to remain idle before the connection is terminated. A setting of **0** means that this function is disabled.	0–7200 seconds	0
maxclients	The maximum number of clients allowed to be concurrently connected to a DCS instance.	1000–10,000	10,000
maxmemory-policy	The policy applied when the maxmemory limit is reached. For more information about data eviction, see **What Is the Default Data Eviction Policy?** or visit the Redis official website.	volatile-lru allkeys-lru volatile-random allkeys-random volatile-ttl noeviction	noeviction
reserved-memory-percent	Percentage of the maximum available memory reserved for background processes, such as data persistence and replication.	0–80	30

Step 8 After you have finished setting the parameters, click **Save**.

Step 9 Click **Yes** to confirm the modification.

----End

Typical Scenarios of Configuring Parameters

The following describes how to change the value of the **appendonly** parameter:

- If Redis is used as the cache and services are insensitive to Redis data losses, disable instance persistence to improve performance. In this case, change the value of **appendonly** to **no**. For details, see **Procedure**.
- If Redis is used as the database or services are sensitive to Redis data losses, enable instance persistence. In this case, change the value of **appendonly** to **yes**. For details, see **Procedure**. After instance persistence is enabled, you need to consider the frequency of writing Redis cache data to disks and the

impact on the Redis performance. You can use this parameter together with the **appendfsync** parameter. There are three modes of calling fsync():

- **no**: fsync() is never called. The OS will flush data when it is ready. This mode offers the highest performance.
- **always**: fsync() is called after every write to the AOF. This mode is very slow, but also very safe.
- **everysec**: fsync() is called once per second, ensuring both data security and performance.

NOTE

Currently, the **appendonly** and **appendfsync** parameters can be modified on the console only for master/standby and Redis 4.0 and later Redis Cluster instances.

6.3 Modifying the Security Group

On the DCS console, after creating a DCS instance, you can modify the security group of the DCS instance on the instance's **Basic Information** page.

You can modify the security groups of DCS Redis 3.0 instances but cannot modify those of DCS Redis 4.0/5.0/6.0 instances.

Prerequisites

At least one DCS instance has been created.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of the DCS instance for which you want to modify the security group.

Step 5 Click the **Basic Information** tab. In the **Network** area, click ✎ next to the **Security Group** parameter.

Step 6 Select a new security group from the drop-down list. Click ✓ to save the modification or ✗ to discard the modification.

NOTE

Only the security groups that have been created can be selected from the drop-down list. If you need to create a security group, follow the procedure described **Security Group Configurations**.

The modification will take effect immediately, that is, the new maintenance time window will appear on the **Basic Information** tab page immediately.

----End

6.4 Viewing Background Tasks

After you initiate certain instance operations such as modifying instance specifications and changing or resetting a password, a background task will start for the operation. On the DCS console, you can view the background task status and clear task information by deleting task records.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of the DCS instance whose background task you want to manage.

Step 5 Click the **Background Tasks** tab.

A list of background tasks is displayed.

Step 6 Click 📅 , specify **Start Date** and **End Date**, and click **OK** to view tasks started in the corresponding time segment.

- Click ⟳ to refresh the task status.
- To clear the record of a background task, click **Delete** in the **Operation** column.

  NOTE

  You can only delete the records of tasks in the **Successful** or **Failed** state.

----End

6.5 Viewing Data Storage Statistics of a DCS Redis 3.0 Proxy Cluster Instance

You can view the data storage statistics of all nodes of a DCS Redis 3.0 Proxy Cluster instance. If data storage is unevenly distributed across nodes, you can scale up the instance or clear data.

You can only view data storage statistics of DCS Redis 3.0 Proxy Cluster instances. Instances of other types, for example, master/standby, only have one node, and you can view the used memory on the instance details page.

NOTE

A Redis Cluster instance has multiple storage nodes. You can check the data storage statistics of a Redis Cluster instance in its Redis Server monitoring data.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Filter DCS instances to find the desired one.

Step 4 Click the name of the desired Proxy Cluster instance to go to the instance details page.

Step 5 Click the **Node Management** tab.

The data volume of each node in the cluster instance is displayed.

When the data storage capacity of a node in a cluster is used up, you can scale up the instance according to **Modifying DCS Instance Specifications**.

----End

6.6 Managing Tags

Tags facilitate DCS instance identification and management.

NOTE

If tag policies for DCS have been set in your organization, add tags to DCS instances based on these policies. If a tag does not comply with the policies, tag addition may fail. Contact your organization administrator to learn more about tag policies.

You can add tags to an instance when creating it or add, modify, or delete tags on the details page of a created instance. Each instance can have a maximum of 20 tags.

A tag consists of a tag key and a tag value. **Table 6-3** lists the tag key and value requirements.

Table 6-3 Tag key and value requirements

Parameter	Requirement
Tag key	• Cannot be left blank. • Must be unique for the same instance. • Cannot start or end with a space. • Consists of a maximum of 128 characters. • Can contain letters of any language, digits, spaces, and special characters _.:=+-@ • Cannot start with _**sys**_.

Parameter	Requirement
Tag value	● Consists of a maximum of 255 characters. ● Can contain letters of any language, digits, spaces, and special characters _.:/=+-@ ● Cannot start or end with a space.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of an instance.

Step 5 Choose **Instance Configuration** > **Tags**.

View the tags of the instance.

Step 6 Perform the following operations as required:

- Add a tag
  a. Click **Add/Edit Tag**.

     If you have created predefined tags, select a predefined pair of tag key and value. To view predefined tags or create tags, click **View predefined tags**. You will be directed to the TMS console.

     You can also create new tags by entering **Tag key** and **Tag value**.

  b. Click **OK**.
- Modify a tag

  Click **Add/Edit Tag**. In the displayed **Add/Edit Tag** dialog box, delete the desired key, add the key again, enter a new tag value, and click **Add**.
- Delete a tag

  In the row containing the tag to be deleted, click **Delete** in the **Operation** column. Then click **Yes**.

----End

6.7 Managing Shards and Replicas

This section describes how to query the shards and replicas of a DCS Redis 4.0/5.0 instance and how to manually promote a replica to master.

Currently, **this function is supported only by master/standby and cluster DCS Redis 4.0/5.0/6.0 instances. DCS Redis 3.0 instances and single-node DCS Redis 4.0/5.0/6.0 instances do not support this function.**

- A master/standby instance has only one shard with one master and one replica by default. You can view the sharding information on the **Shards and Replicas** page. To manually switch the master and replica roles, see **Performing a Master/Standby Switchover for a DCS Instance**.
- A Redis Cluster instance has multiple shards. Each shard has one master and one replica by default. On the **Shards and Replicas** page, you can view the sharding information and manually switch the master and replica roles. For details about the number of shards corresponding to different instance specifications, see **Redis Cluster**.

Promoting a Replica to Master

Step 1 Log in to the DCS console.

Step 2 Click in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**. The **Cache Manager** page is displayed.

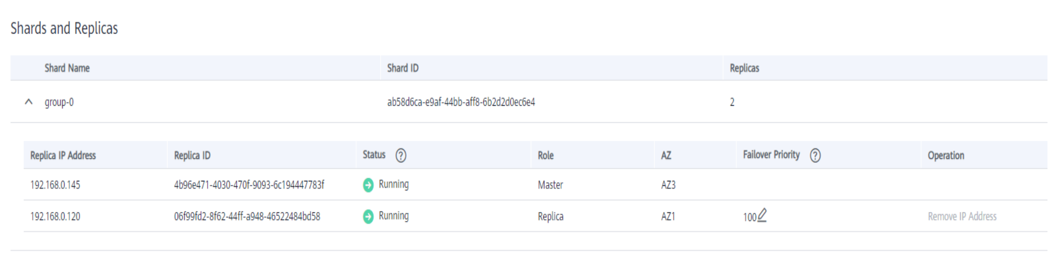
Step 4 Click an instance.

Step 5 Click the **Shards and Replicas** tab.

The page displays all shards in the instance and the list of replicas of each shard.

Step 6 Click to show all replicas of a shard.

Figure 6-1 Lists of shards and replicas



Shards and Replicas

Shard Name	Shard ID	Replicas
group-0	ab58d6ca-e9af-44bb-aff8-6b2d2d0ec6e4	2

Replica IP Address	Replica ID	Status	Role	AZ	Failover Priority	Operation
192.168.0.145	4b96e471-4030-470f-9093-6c194447783f	Running	Master	AZ3		
192.168.0.120	06f99fd2-8f62-44ff-a948-46522484bd58	Running	Replica	AZ1	100	Remove IP Address

- For a cluster instance, you can promote a replica in a shard to be master.
  a. Click **Promote to Master** in the row containing another replica which is in the "Replica" role.
  b. Click **Yes**.
- **Failover Priority** can be set for replicas for a master/standby instance.

  Failover priority: If the master fails, the system will be automatically promoted to the replica with the highest priority you specified. If there are multiple replicas sharing the same priority, a selection and switch process will be performed in the internal system. Priority ranges from 0 to 100 in descending order. **0** indicates that the replica will never be automatically promoted, **1** indicates the highest priority, and **100** indicates the lowest priority.

----End

6.8 Analyzing Big Keys and Hot Keys

By performing big key analysis and hot key analysis, you will have a picture of keys that occupy a large space and keys that are the most frequently accessed.

Notes on big key analysis:

- All DCS Redis instances support big key analysis.
- During big key analysis, all keys will be traversed. The larger the number of keys, the longer the analysis takes.
- Perform big key analysis during off-peak hours and avoid automatic backup periods.
- For a master/standby or cluster instance, the big key analysis is performed on the standby node, so the impact on the instance is minor. For a single-node instance, the big key analysis is performed on the only node of the instance and will reduce the instance access performance by up to 10%. Therefore, perform big key analysis on single-node instances during off-peak hours.
- A maximum of 100 big key analysis records (20 for Strings; 80 for Lists/Sets/Zsets/Hashes and max. 20 for each type) are retained for each instance. When this limit is reached, the oldest records will be deleted to make room for new records. You can also manually delete records you no longer need.

Notes on hot key analysis:

- Only DCS Redis 4.0/5.0/6.0 instances support hot key analysis, and the **maxmemory-policy** parameter of the instances must be set to **allkeys-lfu** or **volatile-lfu**.
- During hot key analysis, all keys will be traversed. The larger the number of keys, the longer the analysis takes.
- Perform hot key analysis shortly after peak hours to ensure the accuracy of the analysis results.
- The hot key analysis is performed on the master node of each instance and will reduce the instance access performance by up to 10%.
- A maximum of 100 hot key analysis records are retained for each instance. When this limit is reached, the oldest records will be deleted to make room for new records. You can also manually delete records you no longer need.

NOTE

Perform big key and hot key analysis during off-peak hours to avoid 100% CPU usage.

Big Key Analysis Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of a DCS Redis instance.

Step 5 Choose **Analysis and Diagnosis** > **Cache Analysis**.

Step 6 On the **Big Key Analysis** tab page, manually perform big key analysis or schedule daily automatic analysis.

Step 7 After an analysis task completes, click **View** to view the analysis results.

You can view the analysis results of different data types.

NOTE

A maximum of 20 big key analysis records are retained for Strings and 80 are retained for Lists, Sets, Zsets, and Hashes.

Table 6-4 Results of big key analysis

Parameter	Description
Key	Name of a big key.
Type	Type of a big key, which can be string, list, set, zset, or hash.
Size	Size or number of elements of a big key.
Shard	Shard where the big key of the cluster instance is located.
Database	Database where a big key is located.

----End

Hot Key Analysis Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of a DCS Redis instance.

Step 5 Choose **Analysis and Diagnosis** > **Cache Analysis**.

Step 6 On the **Hot Key Analysis** tab page, manually perform hot key analysis or schedule daily automatic analysis.

NOTE

If hot key analysis cannot be performed, set **maxmemory-policy** to **allkeys-lfu** or **volatile-lfu**. If this parameter has already been set to **allkeys-lfu** or **volatile-lfu**, perform hot key analysis right away.

Step 7 After an analysis task completes, click **View** to view the analysis results.

The hot key analysis results are displayed.

NOTE

The console displays a maximum of 100 hot key analysis records for each instance.

Table 6-5 Results of hot key analysis

Parameter	Description
Key	Name of a hot key.
Type	Type of a hot key, which can be string, hash, list, set, or sorted set.
Size	Size of the hot key value.
FREQ	Reflects the access frequency of a key within a specific period of time. **FREQ** is the logarithmic access frequency counter. The maximum value of **FREQ** is 255, which indicates 1 million access requests. After **FREQ** reaches **255**, it will no longer increment even if access requests continue to increase. **FREQ** will decrement by 1 for every minute during which the key is not accessed.
Shard	Shard where the hot key of the cluster instance is located.
DataBase	Database where a hot key is located.

----End

6.9 Managing IP Address Whitelist

DCS helps you control access to your DCS instances in the following ways, depending on the deployment mode:

- To control access to DCS Redis 3.0 and Memcached instances, you can use security groups. Whitelists are not supported. For details about how to configure a security group, see **Security Group Configurations**.
- To control access to DCS Redis 4.0/5.0/6.0 instances, you can use whitelists. Security groups are not supported.

The following describes how to manage whitelists of a Redis 4.0/5.0/6.0 instance to allow access only from whitelisted IP addresses. If no whitelists are added for the instance or the whitelist function is disabled, all IP addresses that can communicate with the VPC can access the instance.

Creating a Whitelist Group

Step 1 In the navigation pane, choose **Cache Manager**.

Step 2 Click the name of a DCS instance.

Step 3 Choose **Instance Configuration** > **Whitelist** and then click **Create Whitelist Group**.

Step 4 In the **Create Whitelist Group** dialogue box, specify **Group Name** and **IP Address/Range**.

Table 6-6 Whitelist parameters

Parameter	Description	Example
Group Name	Whitelist group name of the instance. A maximum of four whitelist groups can be created for each instance.	DCS-test
IP Address/ Range	A maximum of 20 IP addresses or IP address ranges can be added to an instance. Separate multiple IP addresses or IP address ranges with commas. Unsupported IP address and IP address range: 0.0.0.0 and 0.0.0.0/0.	10.10.10.1,10.10.10.10

Step 5 Click **OK**.

A whitelist group is automatically enabled for the instance once created. Only whitelisted IP addresses can access the instance.

NOTE

- In the whitelist group list, click **Modify** to modify the IP addresses or IP address ranges in a group, and click **Delete** to delete a whitelist group.
- After whitelist has been enabled, you can click **Disable Whitelist** above the whitelist group list to allow all IP addresses connected to the VPC to access the instance.

----End

6.10 Viewing Redis Slow Queries

Redis logs queries that exceed a specified execution time. You can view the slow query log on the DCS console to identify performance issues.

For details about the commands, visit the **Redis official website**.

Configure the slow log with the following parameters:

- **slowlog-log-slower-than**: The maximum time allowed, in microseconds, for command execution. If this threshold is exceeded, Redis will log the command. The default value is **10,000**. That is, if command execution exceeds 10 ms, the command will be logged.
- **slowlog-max-len**: The maximum allowed number of slow queries that can be logged. The default value is **128**. That is, if the number of slow queries exceeds 128, the earliest record will be deleted to make room for new ones.

For details about the configuration parameters, see **Modifying Configuration Parameters**.

NOTE

You can view the slow log of a Proxy Cluster DCS Redis 3.0 instance only if the instance is created after October 14, 2019. If the instance was created earlier, contact technical support to upgrade it. The upgrade adds the slow log function to the console, and does not affect services.

Viewing Slow Queries on the Console

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of a DCS instance.

Step 5 Choose **Analysis and Diagnosis** > **Slow Queries**.

Step 6 Select a start date and an end date to view slow queries within the specified period.

NOTE

- For details about the commands, visit the **Redis official website**.
- Slow queries of up to seven days can be queried.

----End

6.11 Viewing Redis Run Logs

You can create run log files on the DCS console to collect run logs of DCS Redis instances within a specified period. After the logs are collected, you can download the log files to view the logs.

This function is supported by DCS Redis 4.0 instances and later.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of a DCS instance.

Step 5 Click **Run Logs**.

Step 6 Click **Create Log File**.

For a master/standby or cluster instance, logs will be collected from the specified shard and replica. If the instance is the single-node type, logs of the only node of the instance will be collected.

Select the collection period and click **OK**.

Step 7 After the log file is successfully collected, click **Download** to download it.

----End

6.12 Diagnosing an Instance

Scenario

If a fault or performance issue occurs, you can ask DCS to diagnose your instance to learn about the cause and impact of the issue and how to handle it.

Restrictions

DCS Redis 3.0 and Memcached instances do not support diagnosis.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner of the management console and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of a DCS Redis instance.

Step 5 Choose **Analysis and Diagnosis** > **Instance Diagnosis**.

Step 6 Specify the tested object and time range, and click **Start Diagnosis**.

- **Tested Object**: You can select a single node or all nodes. By default, all nodes are tested.
- **Range**: You can specify up to 10 minutes before a point in time in the last 7 days.

NOTE

Instance diagnosis may fail during specification modification.

Step 7 After the diagnosis is complete, you can view the result in the **Test History** list. If the result is abnormal, click **View Report** for details. You can click **Delete** to delete a record.

In the report, you can view the cause and impact of abnormal items and suggestions for handling them.

----End

6.13 Transmitting DCS Redis Data with Encryption Using SSL

DCS Redis 6.0 instances support SSL encryption to ensure data transmission security. This function is not available for other instance versions. RESP (REdis Serialization Protocol), the communication protocol of Reids, only supports plaintext transmission in versions earlier than Redis 6.0.

Enabling or Disabling SSL

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner of the management console and select the region where your instance is located.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 On the **Cache Manager** page, click a DCS instance.

Step 5 In the navigation pane, choose **SSL**.

Step 6 Click ✎ next to **SSL Certificate** to enable or disable SSL.

NOTICE

- Enabling or disabling SSL will restart the instance and disconnect it for a few seconds. Wait until off-peak hours and ensure that your application can re-connect.
- The restart cannot be undone. For single-node DCS instances and other instances where AOF persistence is disabled (**appendonly** is set to **no**), data will be cleared and ongoing backup tasks will be stopped. Exercise caution when performing this operation.
- Enabling SSL will deteriorate read/write performance.

Step 7 Click **Download Certificate** to download the SSL certificate.

Step 8 Decompress the SSL certificate and upload the decompressed **ca.crt** file to the server where the Redis client is located.

Step 9 Add the path of the **ca.crt** file to the command for connecting to the instance. If you use redis-cli to connect to an instance, refer to **Accessing a DCS Redis Instance Through redis-cli**.

----End

7 Backing Up and Restoring DCS Instances

7.1 Overview

On the DCS console, you can back up and restore DCS instances.

Importance of DCS Instance Backup

There is a small chance that inconsistent data could exist in a DCS instance owing to service system exceptions or problems in loading data from persistence files. In addition, some systems demand not only high reliability but also data security, data restoration, and even permanent data storage.

Currently, data in DCS instances can be backed up to OBS. If a DCS instance becomes faulty, data in the instance can be restored from backup so that service continuity is not affected.

Backup Modes

DCS instances support the following backup modes:

- Automated backup

  You can create a scheduled backup policy on the DCS console. Then, data in the chosen DCS instances will be automatically backed up at the scheduled time.

  You can choose the days of the week on which scheduled backup will run. Backup data will be retained for a maximum of seven days. Backup data older than seven days will be automatically deleted.

  The primary purpose of scheduled backups is to create complete data replicas of DCS instances so that the instance can be quickly restored if necessary.

- Manual backup

  Backup requests can be issued manually. Data in the chosen DCS instances will be backed up to OBS.

  Before performing high-risk operations, such as system maintenance or upgrade, back up DCS instance data.

  When a DCS instance is in use, its backup data will not be automatically deleted. You can manually delete backup data as required. When you delete the instance, its backup data is deleted along with the instance. If you need the backup data, download and save it in advance.

Additional Information About Data Backup

- Instance type
  - Redis: Only master/standby, Proxy Cluster, and Redis Cluster instances can be backed up and restored, while single-node instances cannot. However, you can export data of a single-node instance to an RDB file using redis-cli. For details, see **Can I Export Backup Data of DCS Redis Instances to RDB Files Using the Console?**
  - Memcached: Only master/standby instances can be backed up and restored, while single-node instances cannot.
- Backup mechanisms

  DCS for Redis 3.0 persists data with Redis AOF. DCS for Redis 4.0/5.0/6.0 persist data to RDB or AOF files in manual backup mode, and to RDB files in automatic backup mode.

  Backup tasks run on standby cache nodes. DCS instance data is backed up by compressing and storing the data persistence files from the standby cache node to OBS.

  DCS checks instance backup policies once an hour. If a backup policy is matched, DCS runs a backup task for the corresponding DCS instance.
- Impact on DCS instances during backup

  Backup tasks run on standby cache nodes, without incurring any downtime.

  In the event of full-data synchronization or heavy instance load, it takes a few minutes to complete data synchronization. If instance backup starts before data synchronization is complete, the backup data will be slightly behind the data in the master cache node.

  During instance backup, the standby cache node stops persisting the latest changes to disk files. If new data is written to the master cache node during backup, the backup file will not contain the new data.
- Backup time

  It is advisable to back up instance data during off-peak periods.
- Storage of backup files

  Backup files are stored to OBS.
- Handling exceptions in scheduled backup

  If a scheduled backup task is triggered while the DCS instance is restarting or being scaled up, the scheduled backup task will be run in the next cycle.

  If backing up a DCS instance fails or the backup is postponed because another task is in progress, DCS will try to back up the instance in the next cycle. A maximum of three retries are allowed within a single day.
- Retention period of backup data

  Scheduled backup files are retained for up to seven days. You can configure the retention period. At the end of the retention period, most backup files of the DCS instance will be automatically deleted, but at least one backup file will be retained.

  Manual backup files are retained permanently and need to be manually deleted. When you delete the instance, its backup files will be deleted along with the instance. If you need to retain the backup data, download the backup files in advance.

Data Restoration

- Data restoration process
  a. You can initiate a data restoration request using the DCS console.
  b. DCS obtains the backup file from OBS.
  c. Read/write to the DCS instance is suspended.
  d. The original data persistence file of the master cache node is replaced by the backup file.
  e. The new data persistence file (that is, the backup file) is reloaded.
  f. Data is restored, and the DCS instance starts to provide read/write service again.
- Impact on service systems

  Restoration tasks run on master cache nodes. During restoration, data cannot be written into or read from instances.
- Handling data restoration exceptions

  If a backup file is corrupted, DCS will try to fix the backup file while restoring instance data. If the backup file is successfully fixed, the restoration proceeds. If the backup file cannot be fixed, the master/standby DCS instance will be changed back to the state in which it was before data restoration.

7.2 Configuring an Automatic Backup Policy

On the DCS console, you can configure an automatic backup policy. The system then backs up data in your instances according to the backup policy.

By default, automatic backup is disabled. To enable it, perform the operations described in this section. Single-node instances do not support backup and restoration.

If automatic backup is not required, disable the automatic backup function in the backup policy.

Prerequisites

A master/standby or cluster DCS instance is in the **Running** state.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of the DCS instance to display more details about the DCS instance.

Step 5 On the instance details page, click **Backups & Restorations**.

Step 6 Slide to the right to enable automatic backup. Backup policies will be displayed.

Table 7-1 Parameters in a backup policy

Parameter	Description
Backup Schedule	Day of a week on which data in the chosen DCS instance is automatically backed up. You can select one or multiple days of a week.
Retention Period (days)	The number of days that automatically backed up data is retained. Backup data will be permanently deleted at the end of retention period and cannot be restored. Value range: 1–7.
Start Time	Time at which automatic backup starts. Value: the full hour between 00:00 to 23:00 The DCS checks backup policies once every hour. If the backup start time in a backup policy has arrived, data in the corresponding instance is backed up. **NOTE** Instance backup takes 5 to 30 minutes. The data added or modified during the backup process will not be backed up. To reduce the impact of backup on services, it is recommended that data should be backed up during off-peak periods. Only instances in the **Running** state can be backed up.

Step 7 Click **OK**.

Step 8 Automatic backup starts at the scheduled time. You can view backup records on the current page.

After the backup is complete, click **Download**, **Restore**, or **Delete** next to the backup record as required.

----End

7.3 Manually Backing Up a DCS Instance

You need to manually back up data in DCS instances in a timely manner. This section describes how to manually back up data in master/standby instances using the DCS console.

By default, manually backed up data is permanently retained. If backup data is no longer in use, you can delete it manually.

Prerequisites

At least one master/standby DCS instance is in the **Running** state.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of the DCS instance to display more details about the DCS instance.

Step 5 On the instance details page, click **Backups & Restorations**.

Step 6 Click **Create Backup**.

Step 7 Select a backup file format.

Only DCS Redis 4.0 and later instances support backup file format selection.

Step 8 In the **Create Backup** dialog box, click **OK**.

Information in the **Description** text box cannot exceed 128 bytes.

After the backup is complete, click **Download**, **Restore**, or **Delete** next to the backup record as required.

> **NOTE**
>
> Instance backup takes 10 to 15 minutes. The data added or modified during the backup process will not be backed up.

----End

7.4 Restoring a DCS Instance

On the DCS console, you can restore backup data to a chosen DCS instance.

Prerequisites

- At least one master/standby or cluster DCS instance is in the **Running** state.
- A backup task has been run to back up data in the instance to be restored and the status of the backup task is **Succeeded**.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of the DCS instance to display more details about the DCS instance.

Step 5 On the instance details page, click **Backups & Restorations**.

A list of historical backup tasks is then displayed.

Step 6 Click **Restore** in the same row as the chosen backup task.

Step 7 Click **OK** to start instance restoration.

Information in the **Description** text box cannot exceed 128 bytes.

The **Restoration History** tab page displays the result of the instance restoration task.

NOTE

Instance restoration takes 1 to 30 minutes.

While being restored, DCS instances do not accept data operation requests from clients because existing data is being overwritten by the backup data.

----End

7.5 Downloading a Backup File

Automatically backed up data can be retained for a maximum of 7 days. Manually backed up data is not free of charge and takes space in OBS. Due to these limitations, you are advised to download the RDB and AOF backup files and permanently save them on the local host.

This function is supported only by master/standby and cluster instances, and not by single-node instances. To export the data of a single-node instance to an RDB file, you can use redis-cli. For details, see **How Do I Export DCS Redis Instance Data?**

To export the data of a master/standby or cluster instance, do as follows:

- Redis 3.0: Export the instance data to AOF files by using the DCS console, or to RDB files by running the **redis-cli -h** *{redis_address}* **-p** ***6379*** **[-a** *{password}***]** **--rdb** *{output.rdb}* command by using redis-cli.
- Redis 4.0 and later: Export the instance data to AOF or RDB files by using the DCS console.

Prerequisites

The instance has been backed up and the backup is still valid.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Filter DCS instances to find the desired one.

Step 4 Click the name of the DCS instance to display more details about the DCS instance.

Step 5 On the instance details page, click **Backups & Restorations**.

A list of historical backup tasks is then displayed.

Step 6 Select the historical backup data to be downloaded, and click **Download**.

Step 7 In the displayed, **Download Backup File** dialog box, select either of the following two download methods.

Download methods:

- By URL
  a. Set the URL validity period and click **Query**.
  b. Download the backup file by using the URL list.

  NOTE

  If you choose to copy URLs, use quotation marks to quote the URLs when running the **wget** command in Linux. For example:

  wget 'https://obsEndpoint.com:443/redisdemo.rdb?parm01=value01&parm02=value02'

  This is because the URL contains the special character and (&), which will confuse the **wget** command. Quoting the URL facilitates URL identification.

- By OBS

  Perform the procedure as prompted.

----End

8 Migrating Data with DCS

8.1 Introduction to Migration with DCS

Migration Modes

DCS for Redis supports online migration (in full or incrementally) and backup migration (by importing backup files).

- Backup migration is suitable when the source and target Redis instances are not connected, and the source Redis instance does not support the **SYNC** and **PSYNC** commands. The data source can be an OBS bucket or a Redis instance.
  - Importing data from an OBS bucket: Download the source Redis data and then upload it to an OBS bucket in the same region as the target DCS Redis instance. DCS will read the backup data from the OBS bucket and migrate the data into the target instance.

    This migration mode can be used for migrating data from other Redis vendors or self-hosted Redis to DCS for Redis.
  - Importing data from a Redis instance: Back up the source Redis data and then migrate the backup data to DCS for Redis.
- Migrating data online: If the source and target instances are interconnected and the **SYNC** and **PSYNC** commands are supported in the source instance, data can be migrated online in full or incrementally from the source to the target.

The following table describes data migration modes supported by DCS.

Table 8-1 DCS data migration modes

Migration Mode	Source	Target: DCS		
		Single-node or master/standby	Proxy Cluster	Redis Cluster

Importing backup files	AOF/RDB file	√	√	√
Migrating data online	DCS for Redis: Single-node or master/ standby	√	√	√
	DCS for Redis: Proxy Cluster **NOTE** Proxy Cluster DCS Redis 3.0 instances cannot be used as the source, while Proxy Cluster DCS Redis 4.0 or 5.0 instances can.	√	√	√
	DCS for Redis: Redis Cluster	√	√	√
	Self-hosted Redis: single-node or master/ standby	√	√	√
	Self-hosted Redis: proxy-based cluster	√	√	√
	Self-hosted Redis: Redis Cluster	√	√	√
	Other Redis: single-node or master/ standby	√	√	√
	Other Redis: proxy-based cluster	√	√	√
	Other Redis: Redis Cluster	√	√	√

	NOTE You can migrate data online in full or incrementally from **other cloud Redis** to **DCS for Redis** if they are connected and the **SYNC** and **PSYNC** commands can be run on the source Redis. However, some instances provided by other cloud vendors may fail to be migrated online. In this case, migrate data through backup import or use other migration schemes. **Migration Solution Notes**

NOTE

- **DCS for Redis** refers to Redis instances provided by DCS
- **Self-hosted Redis** refers to self-hosted Redis on the cloud, from other cloud vendors, or in on-premises data centers.
- **Other Redis** refers to Redis services provided by other cloud vendors.
- **√**: Supported. **×**: Not supported.

8.2 Importing Backup Files

8.2.1 Importing Backup Files from an OBS Bucket

Scenario

Use the DCS console to migrate backup data to DCS for Redis.

Simply download the source Redis backup data and then upload the data to an OBS bucket in the same region as the target DCS Redis instance. After you have created a migration task on the DCS console, DCS will read data from the OBS bucket and data will be migrated to the target instance.

.aof, .rbb, .zip, and .tar.gz files can be uploaded to OBS buckets. You can directly upload .aof and .rdb files or compress them into .zip or .tar.gz files before uploading.

Prerequisites

- The OBS bucket must be in the same region as the target DCS Redis instance.
- The data files to be uploaded must be in .aof, .rdb, .zip, or .tar.gz formats. .zip files must contain .aof or .rdb files.
- To migrate data from a single-node or master/standby Redis instance of other cloud vendors, create a backup task and download the backup file.
- To migrate data from a cluster Redis instance of other cloud vendors, download all backup files and upload all of them to the OBS bucket. Each backup file contains data for a shard of the instance.

Step 1: Prepare the Target DCS Redis Instance

- If a DCS Redis instance is not available, create one first. For details, see **Creating a DCS Redis Instance**.
- If a DCS Redis instance is available, you do not need to create a new one. However, you can clear the instance data before the migration.

  - If the target instance is Redis 4.0 and later, clear the data by referring to **Clearing DCS Instance Data**.
  - If the target instance is a DCS Redis 3.0 instance, run the **FLUSHALL** command to clear data.
  - If the target instance data is not cleared before the migration and the source and target instances contain the same key, the key in the target instance will be overwritten by the key in the source instance after the migration.
- Redis is backward compatible. The target instance version must be the same as or later than the source instance version.

Step 2: Create an OBS Bucket and Upload Backup Files

Step 1 Create an OBS bucket.

1. Log in to the OBS Console and click **Create Bucket**.
2. Select a region.

   The OBS bucket must be in the same region as the target DCS Redis instance.
3. Specify **Bucket Name**.

   The bucket name must meet the naming rules specified on the console.
4. Set **Storage Class** to **Standard**, **Warm** or **Cold**.
5. Set **Bucket Policy** to **Private**, **Public Read**, or **Public Read and Write**.
6. Configure default encryption.
7. Click **Create Now**.

Step 2 Upload the backup data files to the OBS bucket by using OBS Browser+.

If the backup file to be uploaded does not exceed 5 GB, upload the file using the OBS console by referring to step **Step 3**.

If the backup file to be uploaded is larger than 5 GB, perform the following steps to upload the file using OBS Browser+.

1. Download OBS Browser+.

   For details, see section "Downloading OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)*.
2. Install OBS Browser+.

   For details, see section "Installing OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)*.
3. Log in to OBS Browser+.

   For details, see section "Logging In to OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)*.
4. Create a bucket.

   For details, see the *OBS Console Operation Guide* > "Getting Started" > "Creating a Bucket".
5. Upload backup data.

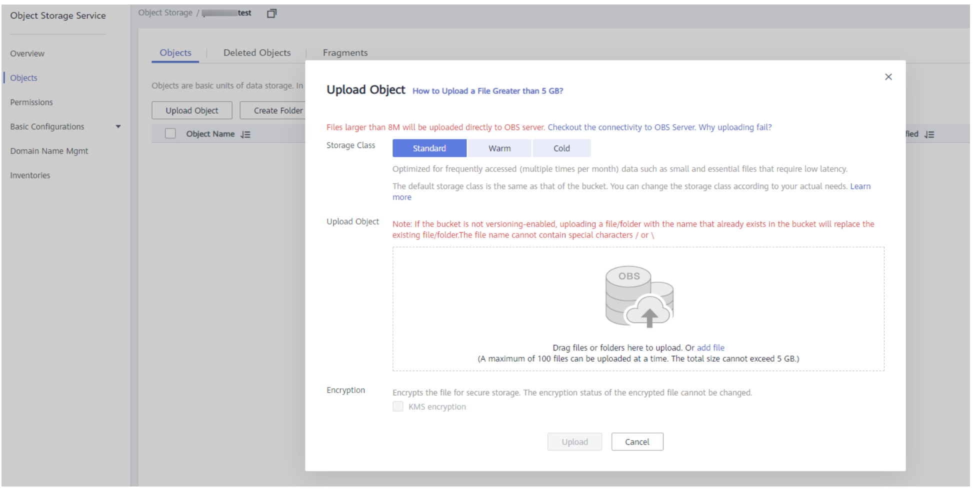
Step 3 On the OBS console, upload the backup data files to the OBS bucket.

Perform the following steps if the backup file size does not exceed 5 GB:

1. In the bucket list, click the name of the created bucket.
2. In the navigation pane, choose **Objects**.
3. On the **Objects** tab page, click **Upload Object**.
4. Upload the objects.

   To upload objects, drag files or folders to the **Upload Object** area or click **add file**. A maximum of 100 files can be uploaded at a time. The total size cannot exceed 5 GB.

   Figure 8-1 Uploading an object

   Objects are basic units of data storage. In

   Files larger than 8M will be uploaded directly to OBS server. Checkout the connectivity to OBS Server. Why uploading fail?

   Optimized for frequently accessed (multiple times per month) data such as small and essential files that require low latency.

   The default storage class is the same as that of the bucket. You can change the storage class according to your actual needs. Learn more

   Note: If the bucket is not versioning-enabled, uploading a file/folder with the name that already exists in the bucket will replace the existing file/folder.The file name cannot contain special characters / or \

   Drag files or folders here to upload. Or add file
   (A maximum of 100 files can be uploaded at a time. The total size cannot exceed 5 GB.)

   Encrypts the file for secure storage. The encryption status of the encrypted file cannot be changed.

   □ KMS encryption

5. (Optional) Select **KMS encryption** to encrypt the file you want to upload.
6. Click **Upload**.

----End

Step 3: Create a Migration Task

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Data Migration**.

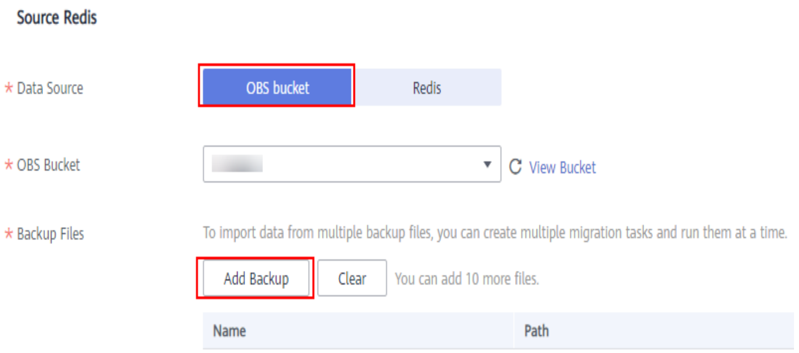
Step 4 Click **Create Backup Import Task**.

Step 5 Specify **Task Name** and **Description**.

Step 6 Select **OBS Bucket** as the data source and then select the OBS bucket to which you have uploaded backup files.

> **NOTE**
>
> You can upload files in the .aof, .rdb, .zip, or .tar.gz format.

Figure 8-2 Importing backup files



Source Redis

* Data Source: OBS bucket | Redis

* OBS Bucket: View Bucket

* Backup Files: To import data from multiple backup files, you can create multiple migration tasks and run them at a time.

Add Backup | Clear | You can add 10 more files.

Name	Path

Step 7 In the **Backup Files** area, click **Add Backup** and select the backup files to be migrated.

Step 8 Select the target DCS Redis instance prepared in **Step 1: Prepare the Target DCS Redis Instance**.

Step 9 Enter the password of the target instance. Click **Test Connection** to verify the password. If the instance is not password-protected, click **Test Connection** directly.

Step 10 Click **Next**.

Step 11 Confirm the migration task details and click **Submit**.

Go back to the data migration task list. After the migration is successful, the task status changes to **Successful**.

----End

8.2.2 Importing Backup Files from Redis

Scenario

Use the DCS console to migrate Redis data from self-hosted Redis to DCS for Redis.

Simply back up your Redis data, create a migration task on the DCS console, and then import the backup to a DCS Redis instance.

Prerequisites

A master/standby or cluster DCS Redis instance has been created as the target for the migration. The source instance has data and has been backed up.

Step 1: Obtain the Source Instance Name and Password

Obtain the name of the source Redis instance.

Step 2: Prepare the Target DCS Redis Instance

- If a DCS Redis instance is not available, create one first. For details, see **Creating a DCS Redis Instance**.
- If a DCS Redis instance is available, you do not need to create a new one. However, you can clear the instance data before the migration.

- If the target instance is Redis 4.0 and later, clear the data by referring to **Clearing DCS Instance Data**.
- If the target instance is a DCS Redis 3.0 instance, run the **FLUSHALL** command to clear data.
- If the target instance data is not cleared before the migration and the source and target instances contain the same key, the key in the target instance will be overwritten by the key in the source instance after the migration.

Step 3: Create a Migration Task

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Data Migration**.

Step 4 Click **Create Backup Import Task**.

Step 5 Enter the task name and description.

Step 6 Set **Data Source** to **Redis**.

Step 7 For source Redis, select the instance prepared in **Step 1: Obtain the Source Instance Name and Password**.

Step 8 Select the backup task whose data is to be migrated.

Step 9 Select the target instance created in **Step 2: Prepare the Target DCS Redis Instance**.

Step 10 Enter the password of the target instance. Click **Test Connection** to verify the password. If the instance is not password-protected, click **Test Connection** directly.

Step 11 Click **Next**.

Step 12 Confirm the migration task details and click **Submit**.

Go back to the data migration task list. After the migration is successful, the task status changes to **Successful**.

----End

8.3 Migrating Data Online

Scenario

If the source and target instances are interconnected and the **SYNC** and **PSYNC** commands are supported in the source instance, data can be migrated online in full or incrementally from the source to the target.

⚠ CAUTION

- If the **SYNC** and **PSYNC** commands are disabled on the source Redis instance, enable them before performing online migration. Otherwise, the migration fails. If you use a DCS Redis instance for online migration, the **SYNC** command is automatically enabled.
- During online migration, you are advised to set **repl-timeout** on the source instance to 300s and **client-output-buffer-limit** to 20% of the maximum memory of the instance.

NOTE

During online migration, results of the **FLUSHDB** and **FLUSHALL** commands executed on the source will not be synchronized to the target.

Impacts on Services

During online migration, data is essentially synchronized in full to a new replica. Therefore, perform online migration during low-demand hours.

Prerequisites

- Before migrating data, read through **Introduction to Migration with DCS** to learn about the DCS data migration function and select an appropriate target instance.
- To migrate data from a single-node or master/standby instance to a cluster instance, check if any data exists in DBs other than DB0 in the source instance. If yes, move the data to DB0 by using the open-source tool Rump. Otherwise, the migration will fail because a cluster instance has only one DB. For details about the migration operations, see **Online Migration with Rump**.

Step 1: Obtain Information About the Source Redis Instance

- If the source is a cloud Redis instance, obtain its name.
- If the source is self-hosted Redis, obtain its IP address or domain name and port number.

Step 2: Prepare the Target DCS Redis Instance

- If a target DCS Redis instance is not available, create one first. For details, see **Creating a DCS Redis Instance**.
- If a target instance is available, you do not need to create a new one. However, you must clear the instance data before the migration. For details, see **Clearing DCS Instance Data**.

  If the target instance data is not cleared before the migration and the source and target instances contain the same key, the key in the target instance will be overwritten by the key in the source instance after the migration.

Step 3: Check the Network

NOTE

- If the source or target of online migration is **Redis in the cloud**, the selected Redis instance must be in the same VPC as the migration task. Otherwise, the migration task may fail to connect to the cloud Redis instance.
- In special scenarios, if you have enabled cross-VPC access between the migration task and the cloud Redis instance, the cloud Redis instance and the migration task can be in different VPCs.

Table 8-2 lists the requirements on the network between the online migration task, source Redis, and target Redis.

Table 8-2 Requirements on the network between the online migration task, source Redis, and target Redis

Source Redis Type	Target Redis Type	Network Requirement on Online Migration
Redis in the cloud	Redis in the cloud	When creating an online migration task, ensure that the online migration task is in the same VPC as the source and target Redis. If they are not in the same VPC, enable cross-network access between the migration task and the source and target Redis. To enable cross-network access, create a VPC peering connection by referring to section "VPC Peering Connection" in *VPC User Guide*.
Redis in the cloud	Self-hosted Redis	When creating an online migration task, ensure that the migration task and the source Redis are in the same VPC. Then, enable cross-network access between the migration task and the target Redis. To enable cross-network access, create a VPC peering connection by referring to section "VPC Peering Connection" in *VPC User Guide*.
Self-hosted Redis	Redis in the cloud	When creating an online migration task, ensure that the migration task and the target Redis are in the same VPC. Then, enable cross-network access between the migration task and the source Redis. To enable cross-network access, create a VPC peering connection by referring to section "VPC Peering Connection" in *VPC User Guide*.
Self-hosted Redis	Self-hosted Redis	After creating an online migration task, enable cross-network access between the migration task and the source and target Redis, respectively. To enable cross-network access, create a VPC peering connection by referring to section "VPC Peering Connection" in *VPC User Guide*.

Step 4: Create a Migration Task

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Data Migration**. The migration task list is displayed.

Step 4 Click **Create Online Migration Task**.

Step 5 Enter the task name and description.

Step 6 Configure the VPC, subnet, and security group for the migration task.

The VPC, subnet, and security group facilitate the migration. Ensure that the migration resources can access the source and target Redis instances.

NOTICE

- The migration task uses a tenant IP address (**Migration ECS** displayed on the **Basic Information** page of the task.) If a whitelist is configured for the source or target instance, add the migration IP address to the whitelist or disable the whitelist.
- To allow the VM used by the migration task to access the source and target instances, set an outbound rule for the task's security group to allow traffic through the IP addresses and ports of the source and target instances. By default, all outbound traffic is allowed.

Step 7 Click **Next**.

Step 8 Click **Submit**.

----End

Configuring the Online Migration Task

Step 1 On the **Online Migration** tab page, click **Configure** in the row containing the online migration task you just created.

Step 2 Specify **Migration Type**.

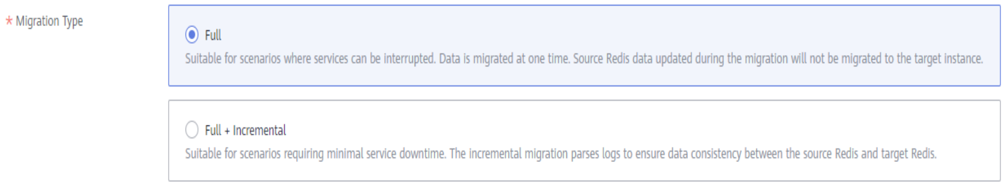
Supported migration types are **Full** and **Full + incremental**, which are described in Table 8-3.

Table 8-3 Migration type description

Migration Type	Description
Full	Suitable for scenarios where services can be interrupted. Data is migrated at one time. Source instance data updated during the migration will not be migrated to the target instance.

Migration Type	Description
Full + incremental	Suitable for scenarios requiring minimal service downtime. The incremental migration parses logs to ensure data consistency between the source and target instances. **Once incremental migration starts, it remains Migrating** until you click **Stop** in the **Operation** column. After the migration is stopped, data in the source instance will not be lost, but data will not be written to the target instance. When the transmission network is stable, the delay of incremental migration is within seconds. The actual delay depends on the transmission quality of the network link.

Figure 8-3 Selecting the migration type



* Migration Type

● Full

Suitable for scenarios where services can be interrupted. Data is migrated at one time. Source Redis data updated during the migration will not be migrated to the target instance.

○ Full + Incremental

Suitable for scenarios requiring minimal service downtime. The incremental migration parses logs to ensure data consistency between the source Redis and target Redis.

Step 3 If **Migration Type** is set to **Full + Incremental**, you can specify a bandwidth limit.

The data synchronization rate can be kept around the bandwidth limit.

Step 4 Specify **Auto-Reconnect**.

If this option is enabled, automatic reconnections will be performed indefinitely in the case of a network exception.

Step 5 Configure source Redis and target Redis.

1. The Redis type can be **Redis in the cloud** or **Self-hosted Redis** as required.
   - **Redis in the cloud**: a DCS Redis instance that is in the same VPC as the migration task
   - **Self-hosted Redis**: self-hosted Redis in another cloud, or in on-premises data centers. If you select this option, enter Redis addresses.

   NOTE

   If the source and target Redis instances are connected but are in different regions of DCS, you can only select **Self-hosted Redis** for **Target Redis Type** and enter the instance addresses, regardless of whether the target Redis instance is self-hosted or in the cloud.
2. If the instance is password-protected, you can click **Test Connection** to check whether the instance password is correct and whether the network is connected. If the instance is not password-protected, click **Test Connection** directly.

Step 6 Click **Next**.

Step 7 Confirm the migration task details and click **Submit**.

Go back to the data migration task list. After the migration is successful, the task status changes to **Successful**.

NOTE

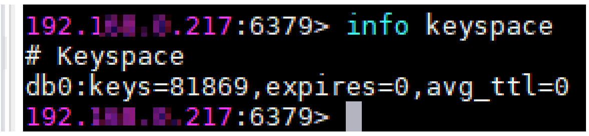
- If the migration type is full+incremental, the migration task status will remain **Migrating** until you click **Stop**.
- After data migration, duplicate keys will be overwritten.

----End

Verifying the Migration

After the migration is complete, use redis-cli to connect the source and target Redis instances to check data integrity.

1. Connect to the source Redis and the target Redis.
2. Run the **info keyspace** command to check the values of **keys** and **expires**.



```
192.1[redacted].[redacted].217:6379> info keyspace
# Keyspace
db0:keys=81869,expires=0,avg_ttl=0
192.1[redacted].[redacted].217:6379>
```

3. Calculate the differences between the values of **keys** and **expires** of the source Redis and the target Redis. If the differences are the same, the data is complete and the migration is successful.

During full migration, source Redis data updated during the migration will not be migrated to the target instance.

8.4 Switching DCS Instance IP Addresses

Scenario

Currently, you cannot change the instance type when using the specification modification function. To modify instance specifications while changing the instance type, you can perform IP switching after data migration. By switching IP addresses, you can also change the AZ used by an instance.

- After online data migration is complete, you can switch the IP addresses.
- The IP addresses can be rolled back as required after the switching.

NOTE

- This function is supported by DCS Redis 4.0 instances and later.
- IP switching is supported only when both the source and target instances are DCS instances in the cloud.

Prerequisites

- Obtain information about the source and target instances. For details about preparing a target instance, see **Step 2: Prepare the Target DCS Redis Instance**.

- Ensure that the source and target instances can communicate with each other. For details, see **Step 3: Check the Network**.
- The target and source instances must use the same port.
- IP switching can be performed only when the following conditions are met:
  - Both the source and target instances are DCS Redis instances.
  - **Table 8-4** lists the supported IP switching scenarios.

Table 8-4 IP switching scenarios

Source	Target
Single-node, master/standby, or Proxy Cluster	Single-node, master/standby, or Proxy Cluster

Precautions for IP Switching

1. Online migration will stop during the switching.
2. Instances will be read-only for one minute and disconnected for several seconds during the switching.
3. The target and source instances must use the same port.
4. If your application cannot reconnect to Redis or handle exceptions, you may need to restart the application after the IP switching.
5. If the source and target instances are in different subnets, the subnet information will be updated after the switching.
6. If the source is a master/standby instance, the IP address of the standby node will not be switched. Ensure that this IP address is not used by your applications.
7. If your applications use a domain name to connect to Redis, the domain name will be used for the source instance. Select **Yes** for **Switch Domain Name**.
8. Ensure that the passwords of the source and target instances are the same. If they are different, verification will fail after the switching.
9. If a whitelist is configured for the source instance, ensure that the same whitelist is configured for the target instance before switching IP addresses.

Switching IP Addresses

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner of the management console and select the region where your instance is located.

Step 3 In the navigation pane, choose **Data Migration**.

Step 4 Click **Create Online Migration Task**.

Step 5 Enter the task name and description.

Step 6 Configure the VPC, subnet, and security group for the migration task.

The VPC, subnet, and security group facilitate the migration. Ensure that the migration resources can access the source and target Redis instances.

Step 7 Configure the migration task by referring to **Configuring the Online Migration Task**. Select **Full + Incremental** for **Migration Type**.

Step 8 On the **Online Migration** page, when the migration task status changes to **Incremental migration in progress**, choose **More** > **Switch IP** in the **Operation** column.

Step 9 In the **Switch IP** dialog box, select whether to switch the domain name.

NOTE

- If a domain name is used, switch it or you must modify the domain name on the client.
- If no domain name is used, the DNS of the instances will be updated.

Step 10 Click **OK**. The IP address switching task is submitted successfully. When the status of the migration task changes to **IP switched**, the IP address switching is complete.

----End

Rolling Back IP Addresses

If you want to change the instance IP address to the original IP address, perform the following operations:

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner of the management console and select the region where your instance is located.

Step 3 In the navigation pane, choose **Data Migration**.

Step 4 On the **Online Migration** page, locate the row that contains the migration task in the **IP switched** state, choose **More** > **Roll Back IP**.

Step 5 In the confirmation dialog box, click **Yes**. The IP address rollback task is submitted successfully. When the task status changes to **IP rolled back**, the rollback is complete.

----End

9 Managing Passwords

9.1 DCS Instance Passwords

Passwords can be configured to control access to your DCS instances, ensuring the security of your data.

You can set an instance access password during or after instance creation. For details on how to set a password after an instance has been created, see **Resetting Instance Passwords**.

You can choose whether to enable password-free access based on your security and convenience trade-off.

NOTE

After 5 consecutive incorrect password attempts, the account for accessing the chosen DCS instance will be locked for 5 minutes. Passwords cannot be changed during the lockout period.

The password must meet the following requirements:

- Cannot be left blank.
- Cannot be the same as the old password.
- Can contain 8 to 32 characters.
- Must contain at least three of the following character types:
  - Lowercase letters
  - Uppercase letters
  - Digits
  - Special characters (`~!@#$^&*()-_=+\|{},<.>/?)

Using Passwords Securely

1. Hide the password when using redis-cli.

   If the **-a <password>** option is used in redis-cli in Linux, the password is prone to leakage because it is logged and kept in the history. You are advised not to use **-a <password>** when running commands in redis-cli. After connecting to Redis, run the **auth** command to complete authentication as shown in the following example:

```
$ redis-cli -h 192.168.0.148 -p 6379
redis 192.168.0.148:6379>auth {yourPassword}
OK
redis 192.168.0.148:6379>
```

2. Use interactive password authentication or switch between users with different permissions.

   If the script involves DCS instance access, use interactive password authentication. To enable automatic script execution, manage the script as another user and authorize execution using sudo.

3. Use an encryption module in your application to encrypt the password.

9.2 Changing Instance Passwords

On the DCS console, you can change the password required for accessing your DCS instance.

NOTE

- You cannot change the password of a DCS instance in password-free mode.
- The DCS instance for which you want to change the password is in the **Running** state.
- The new password takes effect immediately on the server without requiring a restart. The client must reconnect to the server using the new password after a pconnect connection is closed. (The old password can still be used before disconnection.)

Prerequisites

At least one DCS instance has been created.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click ⦿ in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Choose **More** > **Change Password** in the same row as the chosen instance.

Step 5 In the displayed dialog box, set **Old Password**, **New Password**, and **Confirm Password**.

NOTE

After 5 consecutive incorrect password attempts, the account for accessing the chosen DCS instance will be locked for 5 minutes. Passwords cannot be changed during the lockout period.

The password must meet the following requirements:

- Cannot be left blank.
- The new password cannot be the same as the old password.
- Can contain 8 to 32 characters.
- Must contain at least three of the following character types:
  - Lowercase letters
  - Uppercase letters
  - Digits
  - Special characters (`~!@#$^&*()-_=+\|{},<.>/?)

Step 6 In the **Change Password** dialog box, click **OK** to confirm the password change.

----End

9.3 Resetting Instance Passwords

On the DCS console, you can configure a new password if you forget your instance password.

NOTE

- For a DCS Redis or Memcached instance, you can change it from password mode to password-free mode or from password-free mode to password mode by resetting its password. For details, see **Changing Password Settings for DCS Redis Instances** and **Changing Password Settings for DCS Memcached Instances**.
- The DCS instance for which you want to reset the password is in the **Running** state.
- The new password takes effect immediately on the server without requiring a restart. The client must reconnect to the server using the new password after a pconnect connection is closed. (The old password can still be used before disconnection.)

Prerequisites

At least one DCS instance has been created.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Choose **More** > **Reset Password** in the row containing the chosen instance.

Step 5 In the **Reset Password** dialog box, enter a new password and confirm the password.

NOTE

The password must meet the following requirements:

- Cannot be left blank.
- Can contain 8 to 32 characters.
- Contain at least three of the following character types:
  - Lowercase letters
  - Uppercase letters
  - Digits
  - Special characters (`~!@#$^&*()-_=+\|{},<.>/?)

Step 6 Click **OK**.

NOTE

The system will display a success message only after the password is successfully reset on all nodes. If the reset fails, the instance will restart and the password of the cache instance will be restored.

----End

9.4 Changing Password Settings for DCS Redis Instances

Scenario

DCS Redis instances can be accessed with or without passwords. After an instance is created, you can change its password setting in the following scenarios:

- To access a DCS Redis instance in password-free mode, you can enable password-free access to clear the existing password of the instance.

NOTE

- To change the password setting, the DCS Redis instance must be in the **Running** state.
- Password-free access may compromise security. You can set a password by using the password reset function.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 To change the password setting for a DCS Redis instance, choose **Operation** > **More** > **Reset Password** in the same row as the chosen instance.

Step 5 In the **Reset Password** dialogue box, perform either of the following operations as required:

- From password-protected to password-free:

  Switch the toggle for **Password-Free Access** and click **OK**.

- From password-free to password-protected:

  Enter a password, confirm the password, and click **OK**.

----End

9.5 Changing Password Settings for DCS Memcached Instances

Scenario

DCS Memcached instances can be accessed with or without passwords. After an instance is created, you can change its password setting in the following scenarios:

- If you want to access a password-protected DCS Memcached instance without the username and password, you can enable password-free access to clear the username and password of the instance.

  The Memcached text protocol does not support username and password authentication. To access a DCS Memcached instance by using the Memcached text protocol, you must enable password-free access to the instance.
- If you want to access a password-free DCS Memcached instance using a username and password, you can set a password for the instance using the password reset function.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 To enable password-free access to a DCS Memcached instance, choose **Operation** > **More** > **Reset Password** in the same row as the chosen instance.

Step 5 In the **Reset Password** dialogue box, perform either of the following operations as required:

- From password-protected to password-free:

  Switch the toggle for **Password-Free Access** and click **OK**.
- From password-free to password-protected:

  Enter a password, confirm the password, and click **OK**.

----End

10 Parameter Templates

10.1 Viewing Parameter Templates

This section describes how to view parameter templates on the DCS console.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Parameter Templates**.

Step 4 Choose the **Default Templates** or **Custom Templates** tab.

Step 5 View parameter templates.

Currently, you can enter a keyword in the search box to search for a parameter template by template name.

Step 6 Click a parameter template. The parameters contained in the template are displayed. For details about the parameters, see **Table 10-1**.

Table 10-1 DCS Redis instance configuration parameters

Parameter	Description	Value Range	Default Value
active-expire-num	Number of expired keys that can be deleted in regular scans. Redis 3.0 instances do not have this parameter.	1–1000	20

Parameter	Description	Value Range	Default Value
timeout	The maximum amount of time (in seconds) a connection between a client and the DCS instance can be allowed to remain idle before the connection is terminated. A setting of **0** means that this function is disabled. Proxy Cluster instances do not have this parameter.	0–7200 seconds	0
appendfsync	Controls how often fsync() transfers cached data to the disk. Note that some OSs will perform a complete data transfer but some others only make a "best-effort" attempt. There are three settings: no: fsync() is never called. The OS will flush data when it is ready. This mode offers the highest performance. always: fsync() is called after every write to the AOF. This mode is very slow, but also very safe. everysec: fsync() is called once per second. This mode provides a compromise between safety and performance. Single-node instances do not have this parameter.	● no ● always ● everysec	no

Parameter	Description	Value Range	Default Value
appendonly	Indicates whether to log each modification of the instance (that is, data persistence). By default, data is written to disks asynchronously in Redis. If this function is disabled, recently-generated data might be lost in the event of a power failure. Options: yes: enabled no: disabled Single-node instances do not have this parameter.	- yes - no	yes
client-output-buffer-limit-slave-soft-seconds	Number of seconds that the output buffer remains above **client-output-buffer-slave-soft-limit** before the client is disconnected. Single-node instances do not have this parameter.	0–60	60
client-output-buffer-slave-hard-limit	Hard limit (in bytes) on the output buffer of replica clients. Once the output buffer exceeds the hard limit, the client is immediately disconnected. Single-node instances do not have this parameter.	Depends on the instance type and specifications.	Depends on the instance type and specifications.

Parameter	Description	Value Range	Default Value
client-output-buffer-slave-soft-limit	Soft limit (in bytes) on the output buffer of replica clients. Once the output buffer exceeds the soft limit and continuously remains above the limit for the time specified by the **client-output-buffer-limit-slave-soft-seconds** parameter, the client is disconnected. Single-node instances do not have this parameter.	Depends on the instance type and specifications.	Depends on the instance type and specifications.

Parameter	Description	Value Range	Default Value
maxmemory-policy	The deletion policy to apply when the maxmemory limit is reached. Options: **volatile-lru**: Evict keys by trying to remove the less recently used (LRU) keys first, but only among keys that have an expire set. **(Recommended)** **allkeys-lru**: Evict keys by trying to remove the LRU keys first. **volatile-random**: evict keys randomly, but only evict keys with an expire set. **allkeys-random**: Evict keys randomly. **volatile-ttl**: Evict keys with an expire set, and try to evict keys with a shorter time to live (TTL) first. **noeviction**: Do not delete any keys and only return errors when the memory limit was reached. **volatile-lfu**: Evict keys by trying to remove the less frequently used (LFU) keys first, but only among keys that have an expire set. **allkeys-lfu**: Evict keys by trying to remove the LFU keys first.	Depends on the instance version.	Depends on the instance version and type.
lua-time-limit	Maximum time allowed for executing a Lua script (in milliseconds).	100–5000	5,000

Parameter	Description	Value Range	Default Value
master-read-only	Sets the instance to be read-only. All write operations will fail. Proxy Cluster instances do not have this parameter.	● yes ● no	no
maxclients	The maximum number of clients allowed to be concurrently connected to a DCS instance. Proxy Cluster instances do not have this parameter.	Depends on the instance type and specifications.	Depends on the instance type and specifications.
proto-max-bulk-len	Maximum size of a single element request (in bytes).	1,048,576–536,870,912	536,870,912
repl-backlog-size	The replication backlog size (bytes). The backlog is a buffer that accumulates replica data when replicas are disconnected from the master. When a replica reconnects, a partial synchronization is performed to synchronize the data that was missed while replicas were disconnected. Single-node instances do not have this parameter.	16,384–1,073,741,824	1,048,576
repl-backlog-ttl	The amount of time, in seconds, before the backlog buffer is released, starting from the last a replica was disconnected. The value **0** indicates that the backlog is never released. Single-node instances do not have this parameter.	0–604,800	3,600

Parameter	Description	Value Range	Default Value
repl-timeout	Replication timeout (in seconds). Single-node instances do not have this parameter.	30–3,600	60
hash-max-ziplist-entries	Hashes are encoded using a memory efficient data structure when the number of entries in hashes is less than the value of this parameter.	1–10,000	512
hash-max-ziplist-value	Hashes are encoded using a memory efficient data structure when the biggest entry in hashes does not exceed the length threshold indicated by this parameter.	1–10,000	64
set-max-intset-entries	When a set is composed of just strings that happen to be integers in radix 10 in the range of 64 bit signed integers, sets are encoded using a memory efficient data structure.	1–10,000	512
zset-max-ziplist-entries	Sorted sets are encoded using a memory efficient data structure when the number of entries in sorted sets is less than the value of this parameter.	1–10,000	128
zset-max-ziplist-value	Sorted sets are encoded using a memory efficient data structure when the biggest entry in sorted sets does not exceed the length threshold indicated by this parameter.	1–10,000	64

Parameter	Description	Value Range	Default Value
latency-monitor-threshold	Threshold time in latency monitoring. Unit: millisecond. Set to **0**: Latency monitoring is disabled. Set to more than 0: All with at least this many milliseconds of latency will be logged. By running the **LATENCY** command, you can perform operations related to latency monitoring, such as obtaining statistical data, and configuring and enabling latency monitoring. Proxy Cluster instances do not have this parameter.	0–86,400,000 ms	0

Parameter	Description	Value Range	Default Value
notify-keyspace-events	Controls which keyspace events notifications are enabled for. If the value is an empty string, this function is disabled. A combination of different values can be used to enable notifications for multiple event types. Possible values: **K**: Keyspace events, published with the **__keyspace@__** prefix. **E**: Keyevent events, published with __keyevent@__ prefix **g**: Generic commands (non-type specific) such as DEL, EXPIRE, and RENAME **$**: String commands **l**: List commands **s**: Set commands **h**: Hash commands **z**: Sorted set commands **x**: Expired events (events generated every time a key expires) **e**: Evicted events (events generated when a key is evicted for maxmemory) **A**: an alias for "g$lshzxe" The parameter value must contain either **K** or **E**. **A** cannot be used together with any of the characters in "g$lshzxe". For example, the value **Kl** means that Redis will notify Pub/Sub clients about keyspace events and list commands. The value **AKE** means Redis will notify Pub/Sub clients about all events.	See the parameter description.	Ex

Parameter	Description	Value Range	Default Value
	Proxy Cluster instances do not have this parameter.		
slowlog-log-slower-than	Redis records queries that exceed a specified execution time. **slowlog-log-slower-than** is the maximum time allowed, in microseconds, for command execution. If this threshold is exceeded, Redis will record the query.	0–1,000,000	10,000
slowlog-max-len	The maximum allowed number of slow queries that can be logged. Slow query log consumes memory, but you can reclaim this memory by running the **SLOWLOG RESET** command.	0–1000	128

NOTE

1. The default values and value ranges of the **maxclients**, **reserved-memory-percent**, **client-output-buffer-slave-soft-limit**, and **client-output-buffer-slave-hard-limit** parameters are related to the instance specifications. Therefore, these parameters are not displayed in the parameter template.
2. For more information about the parameters described in **Table 10-1**, visit **https://redis.io/topics/memory-optimization**.

----End

10.2 Creating a Custom Parameter Template

You can create custom parameter templates for different cache engine versions and instance types based on service requirements.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Parameter Templates**.

Step 4 Click the **Default Templates** or **Custom Templates** tab to create a template based on a default template or an existing custom template.

- If you select **Default Templates**, click **Customize** in the **Operation** column of the row containing the desired cache engine version.
- If you select **Custom Templates**, click **Copy** in the **Operation** column in the row containing the desired custom template.

Step 5 Specify **Template Name** and **Description**.

NOTE

The template name can contain 4 to 64 characters and must start with a letter or digit. Only letters, digits, hyphens (-), underscores (_), and periods (.) are allowed. The description can be empty.

Step 6 Select **Modifiable parameters**.

Currently, you can enter a keyword in the search box to search for a parameter by parameter name.

Step 7 In the row that contains the parameter to be modified, enter a value in the **Assigned Value** column.

Table 10-2 describes the parameters. In most cases, default values are retained.

Table 10-2 DCS Redis instance configuration parameters

Parameter	Description	Value Range	Default Value
active-expire-num	Number of expired keys that can be deleted in regular scans. Redis 3.0 instances do not have this parameter.	1–1000	20
timeout	The maximum amount of time (in seconds) a connection between a client and the DCS instance can be allowed to remain idle before the connection is terminated. A setting of **0** means that this function is disabled. Proxy Cluster instances do not have this parameter.	0–7200 seconds	0

Parameter	Description	Value Range	Default Value
appendfsync	Controls how often fsync() transfers cached data to the disk. Note that some OSs will perform a complete data transfer but some others only make a "best-effort" attempt. There are three settings: no: fsync() is never called. The OS will flush data when it is ready. This mode offers the highest performance. always: fsync() is called after every write to the AOF. This mode is very slow, but also very safe. everysec: fsync() is called once per second. This mode provides a compromise between safety and performance. Single-node instances do not have this parameter.	● no ● always ● everysec	no
appendonly	Indicates whether to log each modification of the instance (that is, data persistence). By default, data is written to disks asynchronously in Redis. If this function is disabled, recently-generated data might be lost in the event of a power failure. Options: yes: enabled no: disabled Single-node instances do not have this parameter.	● yes ● no	yes

Parameter	Description	Value Range	Default Value
client-output-buffer-limit-slave-soft-seconds	Number of seconds that the output buffer remains above **client-output-buffer-slave-soft-limit** before the client is disconnected. Single-node instances do not have this parameter.	0–60	60
client-output-buffer-slave-hard-limit	Hard limit (in bytes) on the output buffer of replica clients. Once the output buffer exceeds the hard limit, the client is immediately disconnected. Single-node instances do not have this parameter.	Depends on the instance type and specifications.	Depends on the instance type and specifications.
client-output-buffer-slave-soft-limit	Soft limit (in bytes) on the output buffer of replica clients. Once the output buffer exceeds the soft limit and continuously remains above the limit for the time specified by the **client-output-buffer-limit-slave-soft-seconds** parameter, the client is disconnected. Single-node instances do not have this parameter.	Depends on the instance type and specifications.	Depends on the instance type and specifications.

Parameter	Description	Value Range	Default Value
maxmemory-policy	The deletion policy to apply when the maxmemory limit is reached. Options: **volatile-lru**: Evict keys by trying to remove the less recently used (LRU) keys first, but only among keys that have an expire set. **(Recommended)** **allkeys-lru**: Evict keys by trying to remove the LRU keys first. **volatile-random**: evict keys randomly, but only evict keys with an expire set. **allkeys-random**: Evict keys randomly. **volatile-ttl**: Evict keys with an expire set, and try to evict keys with a shorter time to live (TTL) first. **noeviction**: Do not delete any keys and only return errors when the memory limit was reached. **volatile-lfu**: Evict keys by trying to remove the less frequently used (LFU) keys first, but only among keys that have an expire set. **allkeys-lfu**: Evict keys by trying to remove the LFU keys first.	Depends on the instance version.	Depends on the instance version and type.
lua-time-limit	Maximum time allowed for executing a Lua script (in milliseconds).	100–5000	5,000

Parameter	Description	Value Range	Default Value
master-read-only	Sets the instance to be read-only. All write operations will fail. Proxy Cluster instances do not have this parameter.	● yes ● no	no
maxclients	The maximum number of clients allowed to be concurrently connected to a DCS instance. Proxy Cluster instances do not have this parameter.	Depends on the instance type and specifications.	Depends on the instance type and specifications.
proto-max-bulk-len	Maximum size of a single element request (in bytes).	1,048,576–536,870,912	536,870,912
repl-backlog-size	The replication backlog size (bytes). The backlog is a buffer that accumulates replica data when replicas are disconnected from the master. When a replica reconnects, a partial synchronization is performed to synchronize the data that was missed while replicas were disconnected. Single-node instances do not have this parameter.	16,384–1,073,741,824	1,048,576
repl-backlog-ttl	The amount of time, in seconds, before the backlog buffer is released, starting from the last a replica was disconnected. The value **0** indicates that the backlog is never released. Single-node instances do not have this parameter.	0–604,800	3,600

Parameter	Description	Value Range	Default Value
repl-timeout	Replication timeout (in seconds). Single-node instances do not have this parameter.	30–3,600	60
hash-max-ziplist-entries	Hashes are encoded using a memory efficient data structure when the number of entries in hashes is less than the value of this parameter.	1–10,000	512
hash-max-ziplist-value	Hashes are encoded using a memory efficient data structure when the biggest entry in hashes does not exceed the length threshold indicated by this parameter.	1–10,000	64
set-max-intset-entries	When a set is composed of just strings that happen to be integers in radix 10 in the range of 64 bit signed integers, sets are encoded using a memory efficient data structure.	1–10,000	512
zset-max-ziplist-entries	Sorted sets are encoded using a memory efficient data structure when the number of entries in sorted sets is less than the value of this parameter.	1–10,000	128
zset-max-ziplist-value	Sorted sets are encoded using a memory efficient data structure when the biggest entry in sorted sets does not exceed the length threshold indicated by this parameter.	1–10,000	64

Parameter	Description	Value Range	Default Value
latency-monitor-threshold	Threshold time in latency monitoring. Unit: millisecond. Set to **0**: Latency monitoring is disabled. Set to more than 0: All with at least this many milliseconds of latency will be logged. By running the **LATENCY** command, you can perform operations related to latency monitoring, such as obtaining statistical data, and configuring and enabling latency monitoring. Proxy Cluster instances do not have this parameter.	0–86,400,000 ms	0

Parameter	Description	Value Range	Default Value
notify-keyspace-events	Controls which keyspace events notifications are enabled for. If the value is an empty string, this function is disabled. A combination of different values can be used to enable notifications for multiple event types. Possible values: **K**: Keyspace events, published with the __**keyspace@**__ prefix. **E**: Keyevent events, published with __keyevent@__ prefix **g**: Generic commands (non-type specific) such as DEL, EXPIRE, and RENAME **$**: String commands **l**: List commands **s**: Set commands **h**: Hash commands **z**: Sorted set commands **x**: Expired events (events generated every time a key expires) **e**: Evicted events (events generated when a key is evicted for maxmemory) **A**: an alias for "g$lshzxe" The parameter value must contain either **K** or **E**. **A** cannot be used together with any of the characters in "g$lshzxe". For example, the value **Kl** means that Redis will notify Pub/Sub clients about keyspace events and list commands. The value **AKE** means Redis will notify Pub/Sub clients about all events.	See the parameter description.	Ex

Parameter	Description	Value Range	Default Value
	Proxy Cluster instances do not have this parameter.		
slowlog-log-slower-than	Redis records queries that exceed a specified execution time. **slowlog-log-slower-than** is the maximum time allowed, in microseconds, for command execution. If this threshold is exceeded, Redis will record the query.	0–1,000,000	10,000
slowlog-max-len	The maximum allowed number of slow queries that can be logged. Slow query log consumes memory, but you can reclaim this memory by running the **SLOWLOG RESET** command.	0–1000	128

NOTE

1. The default values and value ranges of the **maxclients**, **reserved-memory-percent**, **client-output-buffer-slave-soft-limit**, and **client-output-buffer-slave-hard-limit** parameters are related to the instance specifications. Therefore, these parameters cannot be modified.
2. For more information about the parameters described in **Table 10-2**, visit **https://redis.io/topics/memory-optimization**.
3. The **latency-monitor-threshold** parameter is usually used for fault location. After locating faults based on the latency information collected, change the value of **latency-monitor-threshold** to **0** to avoid unnecessary latency.
4. More about the **notify-keyspace-events** parameter:
   - The parameter setting must contain at least a **K** or **E**.
   - **A** is an alias for "g$lshzxe" and cannot be used together with any of the characters in "g$lshzxe".
   - For example, the value **Kl** means that Redis will notify Pub/Sub clients about keyspace events and list commands. The value **AKE** means Redis will notify Pub/Sub clients about all events.

Step 8 Click **OK**.

----End

10.3 Modifying a Custom Parameter Template

You can modify the name, description, and parameters of a custom parameter template based on service requirements.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Parameter Templates**.

Step 4 Choose the **Custom Templates** tab.

Step 5 You can modify a custom parameter template in either of the following ways:

- Click **Edit** in the **Operation** column.
  a. Change the name or modify the description of a template.
  b. In the **Parameters** area, select **Modifiable parameters**. In the row that contains the parameter to be modified, enter a value in the **Assigned Value** column. **Table 10-3** describes the parameters. In most cases, default values are retained.
  c. Click **OK**.
- Click the name of a custom template. On the displayed page, modify parameters.
  a. Select **Modifiable parameters**. Enter a keyword in the search box to search for a parameter by parameter name.
  b. Click **Modify**.
  c. In the row that contains the parameter to be modified, enter a value in the **Assigned Value** column. **Table 10-3** describes the parameters. In most cases, default values are retained.
  d. Click **Save**.

Table 10-3 DCS Redis instance configuration parameters

Parameter	Description	Value Range	Default Value
active-expire-num	Number of expired keys that can be deleted in regular scans. Redis 3.0 instances do not have this parameter.	1–1000	20

Parameter	Description	Value Range	Default Value
timeout	The maximum amount of time (in seconds) a connection between a client and the DCS instance can be allowed to remain idle before the connection is terminated. A setting of **0** means that this function is disabled. Proxy Cluster instances do not have this parameter.	0–7200 seconds	0
appendfsync	Controls how often fsync() transfers cached data to the disk. Note that some OSs will perform a complete data transfer but some others only make a "best-effort" attempt. There are three settings: no: fsync() is never called. The OS will flush data when it is ready. This mode offers the highest performance. always: fsync() is called after every write to the AOF. This mode is very slow, but also very safe. everysec: fsync() is called once per second. This mode provides a compromise between safety and performance. Single-node instances do not have this parameter.	● no ● always ● everysec	no

Parameter	Description	Value Range	Default Value
appendonly	Indicates whether to log each modification of the instance (that is, data persistence). By default, data is written to disks asynchronously in Redis. If this function is disabled, recently-generated data might be lost in the event of a power failure. Options: yes: enabled no: disabled Single-node instances do not have this parameter.	● yes ● no	yes
client-output-buffer-limit-slave-soft-seconds	Number of seconds that the output buffer remains above **client-output-buffer-slave-soft-limit** before the client is disconnected. Single-node instances do not have this parameter.	0–60	60
client-output-buffer-slave-hard-limit	Hard limit (in bytes) on the output buffer of replica clients. Once the output buffer exceeds the hard limit, the client is immediately disconnected. Single-node instances do not have this parameter.	Depends on the instance type and specifications.	Depends on the instance type and specifications.

Parameter	Description	Value Range	Default Value
client-output-buffer-slave-soft-limit	Soft limit (in bytes) on the output buffer of replica clients. Once the output buffer exceeds the soft limit and continuously remains above the limit for the time specified by the **client-output-buffer-limit-slave-soft-seconds** parameter, the client is disconnected. Single-node instances do not have this parameter.	Depends on the instance type and specifications.	Depends on the instance type and specifications.

Parameter	Description	Value Range	Default Value
maxmemory-policy	The deletion policy to apply when the maxmemory limit is reached. Options: **volatile-lru**: Evict keys by trying to remove the less recently used (LRU) keys first, but only among keys that have an expire set. **(Recommended)** **allkeys-lru**: Evict keys by trying to remove the LRU keys first. **volatile-random**: evict keys randomly, but only evict keys with an expire set. **allkeys-random**: Evict keys randomly. **volatile-ttl**: Evict keys with an expire set, and try to evict keys with a shorter time to live (TTL) first. **noeviction**: Do not delete any keys and only return errors when the memory limit was reached. **volatile-lfu**: Evict keys by trying to remove the less frequently used (LFU) keys first, but only among keys that have an expire set. **allkeys-lfu**: Evict keys by trying to remove the LFU keys first.	Depends on the instance version.	Depends on the instance version and type.
lua-time-limit	Maximum time allowed for executing a Lua script (in milliseconds).	100–5000	5,000

Parameter	Description	Value Range	Default Value
master-read-only	Sets the instance to be read-only. All write operations will fail. Proxy Cluster instances do not have this parameter.	● yes ● no	no
maxclients	The maximum number of clients allowed to be concurrently connected to a DCS instance. Proxy Cluster instances do not have this parameter.	Depends on the instance type and specifications.	Depends on the instance type and specifications.
proto-max-bulk-len	Maximum size of a single element request (in bytes).	1,048,576–536,870,912	536,870,912
repl-backlog-size	The replication backlog size (bytes). The backlog is a buffer that accumulates replica data when replicas are disconnected from the master. When a replica reconnects, a partial synchronization is performed to synchronize the data that was missed while replicas were disconnected. Single-node instances do not have this parameter.	16,384–1,073,741,824	1,048,576
repl-backlog-ttl	The amount of time, in seconds, before the backlog buffer is released, starting from the last a replica was disconnected. The value **0** indicates that the backlog is never released. Single-node instances do not have this parameter.	0–604,800	3,600

Parameter	Description	Value Range	Default Value
repl-timeout	Replication timeout (in seconds). Single-node instances do not have this parameter.	30–3,600	60
hash-max-ziplist-entries	Hashes are encoded using a memory efficient data structure when the number of entries in hashes is less than the value of this parameter.	1–10,000	512
hash-max-ziplist-value	Hashes are encoded using a memory efficient data structure when the biggest entry in hashes does not exceed the length threshold indicated by this parameter.	1–10,000	64
set-max-intset-entries	When a set is composed of just strings that happen to be integers in radix 10 in the range of 64 bit signed integers, sets are encoded using a memory efficient data structure.	1–10,000	512
zset-max-ziplist-entries	Sorted sets are encoded using a memory efficient data structure when the number of entries in sorted sets is less than the value of this parameter.	1–10,000	128
zset-max-ziplist-value	Sorted sets are encoded using a memory efficient data structure when the biggest entry in sorted sets does not exceed the length threshold indicated by this parameter.	1–10,000	64

Parameter	Description	Value Range	Default Value
latency-monitor-threshold	Threshold time in latency monitoring. Unit: millisecond. Set to **0**: Latency monitoring is disabled. Set to more than 0: All with at least this many milliseconds of latency will be logged. By running the **LATENCY** command, you can perform operations related to latency monitoring, such as obtaining statistical data, and configuring and enabling latency monitoring. Proxy Cluster instances do not have this parameter.	0–86,400,000 ms	0

Parameter	Description	Value Range	Default Value
notify-keyspace-events	Controls which keyspace events notifications are enabled for. If the value is an empty string, this function is disabled. A combination of different values can be used to enable notifications for multiple event types. Possible values: **K**: Keyspace events, published with the **__keyspace@__** prefix. **E**: Keyevent events, published with __keyevent@__ prefix **g**: Generic commands (non-type specific) such as DEL, EXPIRE, and RENAME **$**: String commands **l**: List commands **s**: Set commands **h**: Hash commands **z**: Sorted set commands **x**: Expired events (events generated every time a key expires) **e**: Evicted events (events generated when a key is evicted for maxmemory) **A**: an alias for "g$lshzxe" The parameter value must contain either **K** or **E**. **A** cannot be used together with any of the characters in "g$lshzxe". For example, the value **Kl** means that Redis will notify Pub/Sub clients about keyspace events and list commands. The value **AKE** means Redis will notify Pub/Sub clients about all events.	See the parameter description.	Ex

Parameter	Description	Value Range	Default Value
	Proxy Cluster instances do not have this parameter.		
slowlog-log-slower-than	Redis records queries that exceed a specified execution time. **slowlog-log-slower-than** is the maximum time allowed, in microseconds, for command execution. If this threshold is exceeded, Redis will record the query.	0–1,000,000	10,000
slowlog-max-len	The maximum allowed number of slow queries that can be logged. Slow query log consumes memory, but you can reclaim this memory by running the **SLOWLOG RESET** command.	0–1000	128

NOTE

1. The default values and value ranges of the **maxclients**, **reserved-memory-percent**, **client-output-buffer-slave-soft-limit**, and **client-output-buffer-slave-hard-limit** parameters are related to the instance specifications. Therefore, these parameters cannot be modified.
2. For more information about the parameters described in **Table 10-3**, visit **https://redis.io/topics/memory-optimization**.
3. The **latency-monitor-threshold** parameter is usually used for fault location. After locating faults based on the latency information collected, change the value of **latency-monitor-threshold** to **0** to avoid unnecessary latency.
4. More about the **notify-keyspace-events** parameter:
   - The parameter setting must contain at least a **K** or **E**.
   - **A** is an alias for "g$lshzxe" and cannot be used together with any of the characters in "g$lshzxe".
   - For example, the value **Kl** means that Redis will notify Pub/Sub clients about keyspace events and list commands. The value **AKE** means Redis will notify Pub/Sub clients about all events.

----End

10.4 Deleting a Custom Parameter Template

This section describes how to delete a custom parameter template.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Parameter Templates**.

Step 4 Choose the **Custom Templates** tab.

Step 5 Click **Delete** in the **Operation** column.

Step 6 Click **Yes**.

----End

11 Monitoring

11.1 DCS Metrics

Introduction

This section describes DCS metrics reported to Cloud Eye as well as their namespaces and dimensions. You can use the Cloud Eye console or call APIs to query the DCS metrics and alarms.

Table 11-1 Monitoring dimensions for different instance types

Instance Type	Instance Monitoring	Redis Server Monitoring	Proxy Monitoring
Single-node	Supported The monitoring on the instance dimension is conducted on the Redis Server.	N/A	N/A
Master/standby	Supported The master node is monitored.	Supported The master and standby nodes are monitored.	N/A
Proxy Cluster	Supported The monitoring data is the aggregated master node data.	Supported Each shard is monitored.	Supported Each proxy is monitored.

Instance Type	Instance Monitoring	Redis Server Monitoring	Proxy Monitoring
Redis Cluster	Supported The monitoring data is the aggregated master node data.	Supported Each shard is monitored.	N/A

Namespace

SYS.DCS

DCS Redis 3.0 Instance Metrics

NOTE

- The **Monitored Object** column lists instances that support the corresponding metrics.
- **Dimensions** lists the metric dimensions.

Table 11-2 DCS Redis 3.0 instance metrics

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	The monitored object's maximum CPU usage among multiple sampling values in a monitoring period Unit: %	0–100%	Single-node, master/standby, or cluster DCS Redis instance	1 minute
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	Single-node, master/standby, or cluster DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
net_in_throughput	Network Input Throughput	Inbound throughput per second on a port Unit: byte/s	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
net_out_throughput	Network Output Throughput	Outbound throughput per second on a port Unit: byte/s	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
node_status	Instance Node Status	Status of instance nodes. If the status is normal, the value is **0**. If the status is abnormal, the value is **1**.	-	Single-node, master/standby, or cluster DCS Redis instance	1 minute
connected_clients	Connected Clients	Number of connected clients (excluding those from slave nodes)	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
client_longest_out_list	Client Longest Output List	Longest output list among current client connections	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
client_biggest_in_buf	Client Biggest Input Buf	Maximum input data length among current client connections Unit: byte	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
blocked_clients	Blocked Clients	Number of clients suspended by block operations such as BLPOP, BRPOP, and BRPOPLPUSH	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
used_memory	Used Memory	Number of bytes used by the Redis server Unit: byte	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
used_memory_rss	Used Memory RSS	Resident set size (RSS) memory that the Redis server has used, which is the memory that actually resides in the memory, including all stack and heap memory but not swapped-out memory Unit: byte	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
used_memory_peak	Used Memory Peak	Peak memory consumed by Redis since the Redis server last started Unit: byte	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
used_memory_lua	Used Memory Lua	Number of bytes used by the Lua engine Unit: byte	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
memory_frag_ratio	Memory Fragmentation Ratio	Current memory fragmentation, which is the ratio between **used_memory_rss**/ **used_memory**.	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
total_connections_received	New Connections	Number of connections received during the monitoring period	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
total_commands_processed	Commands Processed	Number of commands processed during the monitoring period	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
instantaneous_ops	Ops per Second	Number of commands processed per second	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
total_net_input_bytes	Network Input Bytes	Number of bytes received during the monitoring period Unit: byte	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
total_net_output_bytes	Network Output Bytes	Number of bytes sent during the monitoring period Unit: byte	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
instantaneous_input_kbps	Input Flow	Instantaneous input traffic Unit: kbit/s	≥ 0 kbits/s	Single-node, master/standby, or cluster DCS Redis instance	1 minute
instantaneous_output_kbps	Output Flow	Instantaneous output traffic Unit: kbit/s	≥ 0 kbits/s	Single-node, master/standby, or cluster DCS Redis instance	1 minute
rejected_connections	Rejected Connections	Number of connections that have exceeded maxclients and been rejected during the monitoring period	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
expired_keys	Expired Keys	Number of keys that have expired and been deleted during the monitoring period	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
evicted_keys	Evicted Keys	Number of keys that have been evicted and deleted during the monitoring period	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
keyspace_hits	Keyspace Hits	Number of successful lookups of keys in the main dictionary during the monitoring period	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
keyspace_misses	Keyspace Misses	Number of failed lookups of keys in the main dictionary during the monitoring period	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
pubsub_channels	PubSub Channels	Number of Pub/Sub channels	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute
pubsub_patterns	PubSub Patterns	Number of Pub/Sub patterns	≥ 0	Single-node, master/standby, or cluster DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
keyspace_hits_perc	Hit Rate	Ratio of the number of Redis cache hits to the number of lookups. Calculation: keyspace_hits/ (keyspace_hits + keyspace_misses) Unit: %	0–100%	Single-node, master/ standby, or cluster DCS Redis instance	1 minute
command_max_delay	Maximum Command Latency	Maximum latency of commands Unit: ms	≥ 0 ms	Single-node, master/ standby, or cluster DCS Redis instance	1 minute
auth_errors	Authentication Failures	Number of failed authentications	≥ 0	Single-node or master/standby DCS Redis instance	1 minute
is_slow_log_exist	Slow Query Logs	Existence of slow query logs in the instance **NOTE** Slow queries caused by the **MIGRATE**, **SLAVEOF**, **CONFIG**, **BGSAVE**, and **BGREWRITEAOF** commands are not counted.	● **1**: yes ● **0**: no	Single-node or master/standby DCS Redis instance	1 minute
keys	Keys	Number of keys in Redis	≥ 0	Single-node or master/standby DCS Redis instance	1 minute

DCS Redis 4.0/5.0/6.0 Instance Metrics

NOTE

- The **Monitored Object** column lists instances that support the corresponding metrics.
- **Dimensions** lists the metric dimensions.

Table 11-3 DCS Redis 4.0/5.0/6.0 instance metrics

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	The monitored object's maximum CPU usage among multiple sampling values in a monitoring period Unit: %	0–100%	Single-node or master/standby DCS Redis instance	1 minute
command_max_delay	Maximum Command Latency	Maximum latency of commands Unit: ms	≥ 0 ms	DCS Redis instance	1 minute
total_connections_received	New Connections	Number of connections received during the monitoring period	≥ 0	DCS Redis instance	1 minute
is_slow_log_exist	Slow Query Logs	Existence of slow query logs in the instance **NOTE** Slow queries caused by the **MIGRATE**, **SLAVEOF**, **CONFIG**, **BGSAVE**, and **BGREWRITEAOF** commands are not counted.	- **1**: yes - **0**: no	DCS Redis instance	1 minute
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
expires	Keys With an Expiration	Number of keys with an expiration in Redis	≥ 0	DCS Redis instance	1 minute
keyspace_hits_perc	Hit Rate	Ratio of the number of Redis cache hits to the number of lookups. Calculation: keyspace_hits/(keyspace_hits + keyspace_misses) Unit: %	0–100%	DCS Redis instance	1 minute
used_memory	Used Memory	Number of bytes used by the Redis server Unit: KB, MB, or byte (configurable on the console)	≥ 0	DCS Redis instance	1 minute
used_memory_dataset	Used Memory Dataset	Dataset memory that the Redis server has used Unit: KB, MB, or byte (configurable on the console)	≥ 0	DCS Redis instance	1 minute
used_memory_dataset_perc	Used Memory Dataset Ratio	Percentage of dataset memory that server has used Unit: %	0–100%	DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
used_memory_rss	Used Memory RSS	Resident set size (RSS) memory that the Redis server has used, which is the memory that actually resides in the memory, including all stack and heap memory but not swapped-out memory Unit: KB, MB, or byte (configurable on the console)	≥ 0	DCS Redis instance	1 minute
instantaneous_ops	Ops per Second	Number of commands processed per second	≥ 0	DCS Redis instance	1 minute
keyspace_misses	Keyspace Misses	Number of failed lookups of keys in the main dictionary during the monitoring period	≥ 0	DCS Redis instance	1 minute
keys	Keys	Number of keys in Redis	≥ 0	DCS Redis instance	1 minute
rx_controlled	Flow Control Times	Number of flow control times during the monitoring period	≥ 0	DCS Redis instance	1 minute
bandwidth_usage	Bandwidth Usage	Percentage of the maximum bandwidth limit used (the average value of the sum of input and output flows) Unit: %	≥ 0	DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
connections_usage	Connection Usage	Percentage of the current number of connections to the maximum allowed number of connections Unit: %	≥ 0	DCS Redis instance	1 minute
Instance Node Status	Instance Node Status	Status of instance nodes. If the status is normal, the value is **0**. If the status is abnormal, the value is **1**.	-	DCS Redis instance	1 minute
command_max_rt	Maximum Latency	Maximum delay from when the node receives commands to when it responds Unit: μs	≥ 0	Single-node DCS Redis instance	1 minute
command_avg_rt	Average Latency	Average delay from when the node receives commands to when it responds Unit: μs	≥ 0	Single-node DCS Redis instance	1 minute
cpu_avg_usage	Average CPU Usage	Current average usage of CPU resources Unit: %	≥ 0	Single-node or master/standby DCS Redis instance	1 minute
blocked_clients	Blocked Clients	Number of clients suspended by block operations	≥ 0	DCS Redis instance	1 minute
connected_clients	Connected Clients	Number of connected clients (excluding those from slave nodes)	≥ 0	DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
del	DEL	Number of **DEL** commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
evicted_keys	Evicted Keys	Number of keys that have been evicted and deleted during the monitoring period	≥ 0	DCS Redis instance	1 minute
expire	EXPIRE	Number of **EXPIRE** commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
expired_keys	Expired Keys	Number of keys that have expired and been deleted during the monitoring period	≥ 0	DCS Redis instance	1 minute
get	GET	Number of **GET** commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
hdel	HDEL	Number of **HDEL** commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
hget	HGET	Number of **HGET** commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
hmget	HMGET	Number of **HMGET** commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
hmset	HMSET	Number of **HMSET** commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
hset	HSET	Number of **HSET** commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
instantaneous_input_kbps	Input Flow	Instantaneous input traffic Unit: KB/s	≥ 0 KB/s	DCS Redis instance	1 minute
instantaneous_output_kbps	Output Flow	Instantaneous output traffic Unit: KB/s	≥ 0 KB/s	DCS Redis instance	1 minute
memory_frag_ratio	Memory Fragmentation Ratio	Ratio between Used Memory RSS and Used Memory	≥ 0	DCS Redis instance	1 minute
mget	MGET	Number of **MGET** commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
mset	MSET	Number of **MSET** commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
pubsub_channels	PubSub Channels	Number of Pub/Sub channels	≥ 0	DCS Redis instance	1 minute
pubsub_patterns	PubSub Patterns	Number of Pub/Sub patterns	≥ 0	DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
set	SET	Number of **SET** commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
used_memory_lua	Used Memory Lua	Number of bytes used by the Lua engine Unit: KB, MB, or byte (configurable on the console)	≥ 0	DCS Redis instance	1 minute
used_memory_peak	Used Memory Peak	Peak memory consumed by Redis since the Redis server last started Unit: KB, MB, or byte (configurable on the console)	≥ 0	DCS Redis instance	1 minute
sadd	SADD	Number of **SADD** commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
smembers	SMEMBERS	Number of **SMEMBERS** commands processed per second Unit: count/s	0–500,000	DCS Redis instance	1 minute
keyspace_misses	Keyspace Misses	Number of failed lookups of keys in the main dictionary during the monitoring period	≥ 0	DCS Redis instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
used_memory_dataset	Used Memory Dataset	Dataset memory that the Redis server has used Unit: KB, MB, or byte (configurable on the console)	≥ 0	DCS Redis instance	1 minute
used_memory_dataset_perc	Used Memory Dataset Ratio	Percentage of dataset memory that server has used Unit: %	0–100%	DCS Redis instance	1 minute

Redis Server Metrics of DCS Redis Instances

NOTE

- The **Monitored Object** column lists instances that support the corresponding metrics.
- **Dimensions** lists the metric dimensions.

Table 11-4 Redis Server metrics

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	The monitored object's maximum CPU usage among multiple sampling values in a monitoring period Unit: %	0–100%	Redis Server of a cluster instance Redis Server of a master/ standby DCS Redis 4.0/5.0/6.0 instance	1 minute
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	Redis Server of a cluster instance Redis Server of a master/ standby DCS Redis 4.0/5.0/6.0 instance	1 minute
connected_clients	Connected Clients	Number of connected clients (excluding those from slave nodes)	≥ 0	Redis Server of a cluster instance Redis Server of a master/ standby DCS Redis 4.0/5.0/6.0 instance	1 minute
client_longest_out_list	Client Longest Output List	Longest output list among current client connections	≥ 0	Redis Server of a master/ standby or cluster DCS Redis 4.0 instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
client_biggest_in_buf	Client Biggest Input Buf	Maximum input data length among current client connections Unit: byte	≥ 0	Redis Server of a master/standby or cluster DCS Redis 4.0 instance	1 minute
blocked_clients	Blocked Clients	Number of clients suspended by block operations such as BLPOP, BRPOP, and BRPOPLPUSH	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
used_memory	Used Memory	Number of bytes used by the Redis server Unit: byte	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
used_memory_rss	Used Memory RSS	RSS memory that the Redis server has used, which including all stack and heap memory but not swapped-out memory Unit: byte	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
used_memory_peak	Used Memory Peak	Peak memory consumed by Redis since the Redis server last started Unit: byte	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
used_memory_lua	Used Memory Lua	Number of bytes used by the Lua engine Unit: byte	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
memory_frag_ratio	Memory Fragmentation Ratio	Current memory fragmentation, which is the ratio between **used_memory_rss**/ **used_memory**.	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
total_connections_received	New Connections	Number of connections received during the monitoring period	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
total_commands_processed	Commands Processed	Number of commands processed during the monitoring period	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
instantaneous_ops	Ops per Second	Number of commands processed per second	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
total_net_input_bytes	Network Input Bytes	Number of bytes received during the monitoring period Unit: byte	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
total_net_output_bytes	Network Output Bytes	Number of bytes sent during the monitoring period Unit: byte	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
instantaneous_input_kbps	Input Flow	Instantaneous input traffic Unit: KB/s	≥ 0 KB/s	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
instantaneous_output_kbps	Output Flow	Instantaneous output traffic Unit: KB/s	≥ 0 KB/s	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
rejected_connections	Rejected Connections	Number of connections that have exceeded maxclients and been rejected during the monitoring period	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
expired_keys	Expired Keys	Number of keys that have expired and been deleted during the monitoring period	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
evicted_keys	Evicted Keys	Number of keys that have been evicted and deleted during the monitoring period	≥ 0	Redis Server of a cluster instance Redis Server of a master/ standby DCS Redis 4.0/5.0/6.0 instance	1 minute
pubsub_channels	PubSub Channels	Number of Pub/Sub channels	≥ 0	Redis Server of a cluster instance Redis Server of a master/ standby DCS Redis 4.0/5.0/6.0 instance	1 minute
pubsub_patterns	PubSub Patterns	Number of Pub/Sub patterns	≥ 0	Redis Server of a cluster instance Redis Server of a master/ standby DCS Redis 4.0/5.0/6.0 instance	1 minute
keyspace_hits_perc	Hit Rate	Ratio of the number of Redis cache hits to the number of lookups. Calculation: keyspace_hits/ (keyspace_hits + keyspace_misses) Unit: %	0–100%	Redis Server of a cluster instance Redis Server of a master/ standby DCS Redis 4.0/5.0/6.0 instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
command_max_delay	Maximum Command Latency	Maximum latency of commands Unit: ms	≥ 0 ms	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
is_slow_log_exist	Slow Query Logs	Existence of slow query logs in the node **NOTE** Slow queries caused by the **MIGRATE**, **SLAVEOF**, **CONFIG**, **BGSAVE**, and **BGREWRITEAOF** commands are not counted.	● **1**: yes ● **0**: no	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
keys	Keys	Number of keys in Redis	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
sadd	SADD	Number of **SADD** commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
smembers	SMEMBERS	Number of **SMEMBERS** commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
ms_repl_offset	Replication Gap	Data synchronization gap between the master and the replica	-	Replica of a cluster DCS Redis 4.0 or 5.0 instance	1 minute
del	DEL	Number of **DEL** commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
expire	EXPIRE	Number of **EXPIRE** commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
get	GET	Number of **GET** commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
hdel	HDEL	Number of **HDEL** commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
hget	HGET	Number of **HGET** commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
hmget	HMGET	Number of **HMGET** commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
hmset	HMSET	Number of **HMSET** commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
hset	HSET	Number of **HSET** commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
mget	MGET	Number of **MGET** commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
mset	MSET	Number of **MSET** commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
set	SET	Number of **SET** commands processed per second Unit: count/s	0–500,000	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
rx_controlled	Flow Control Times	Number of flow control times during the monitoring period Unit: count	≥ 0	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute
bandwidth_usage	Bandwidth Usage	Percentage of the used bandwidth to the maximum bandwidth limit	0–200%	Redis Server of a cluster instance Redis Server of a master/standby DCS Redis 4.0/5.0/6.0 instance	1 minute

Proxy Metrics

NOTE

- The **Monitored Object** column lists instances that support the corresponding metrics.
- **Dimensions** lists the metric dimensions.

Table 11-5 Proxy metrics of Proxy Cluster DCS Redis 3.0 instances

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	The monitored object's maximum CPU usage among multiple sampling values in a monitoring period Unit: %	0–100%	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
p_connected_clients	Connected Clients	Number of connected clients	≥ 0	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
max_rxpck_per_sec	Max. NIC Data Packet Receive Rate	Maximum number of data packets received by the proxy NIC per second during the monitoring period Unit: packages/second	0–10,000,000	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
max_txpck_per_sec	Max. NIC Data Packet Transmit Rate	Maximum number of data packets transmitted by the proxy NIC per second during the monitoring period Unit: packages/second	0–10,000,000	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
max_rxkB_per_sec	Maximum Inbound Bandwidth	Largest volume of data received by the proxy NIC per second Unit: KB/s	≥ 0 KB/s	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
max_txkB_per_sec	Maximum Outbound Bandwidth	Largest volume of data transmitted by the proxy NIC per second Unit: KB/s	≥ 0 KB/s	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
avg_rxpck_per_sec	Average NIC Data Packet Receive Rate	Average number of data packets received by the proxy NIC per second during the monitoring period Unit: packages/second	0–10,000,000	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
avg_txpck_per_sec	Average NIC Data Packet Transmit Rate	Average number of data packets transmitted by the proxy NIC per second during the monitoring period Unit: packages/second	0–10,000,000	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute
avg_rxkB_per_sec	Average Inbound Bandwidth	Average volume of data received by the proxy NIC per second Unit: KB/s	≥ 0 KB/s	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
avg_txkB_per_sec	Average Outbound Bandwidth	Average volume of data transmitted by the proxy NIC per second Unit: KB/s	≥ 0 KB/s	Proxy in a Proxy Cluster DCS Redis 3.0 instance	1 minute

DCS Memcached Instance Metrics

NOTE

- The **Monitored Object** column lists instances that support the corresponding metrics.
- **Dimensions** lists the metric dimensions.

Table 11-6 DCS Memcached instance metrics

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	The monitored object's maximum CPU usage among multiple sampling values in a monitoring period Unit: %	0–100%	DCS Memcached instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	DCS Memcached instance	1 minute
net_in_throughput	Network Input Throughput	Inbound throughput per second on a port Unit: byte/s	≥ 0	DCS Memcached instance	1 minute
net_out_throughput	Network Output Throughput	Outbound throughput per second on a port Unit: byte/s	≥ 0	DCS Memcached instance	1 minute
mc_connected_clients	Connected Clients	Number of connected clients (excluding those from slave nodes)	≥ 0	DCS Memcached instance	1 minute
mc_used_memory	Used Memory	Number of bytes used by Memcached Unit: byte	≥ 0	DCS Memcached instance	1 minute
mc_used_memory_rss	Used Memory RSS	RSS memory used is the memory that actually resides in the memory, including all stack and heap memory but not swapped-out memory Unit: byte	≥ 0	DCS Memcached instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
mc_used_memory_peak	Used Memory Peak	Peak memory consumed since the server last started Unit: byte	≥ 0	DCS Memcached instance	1 minute
mc_memory_frag_ratio	Memory Fragmentation Ratio	Current memory fragmentation, which is the ratio between **used_memory_rss/used_memory**.	≥ 0	DCS Memcached instance	1 minute
mc_connections_received	New Connections	Number of connections received during the monitoring period	≥ 0	DCS Memcached instance	1 minute
mc_commands_processed	Commands Processed	Number of commands processed during the monitoring period	≥ 0	DCS Memcached instance	1 minute
mc_instantaneous_ops	Ops per Second	Number of commands processed per second	≥ 0	DCS Memcached instance	1 minute
mc_net_input_bytes	Network Input Bytes	Number of bytes received during the monitoring period Unit: byte	≥ 0	DCS Memcached instance	1 minute
mc_net_output_bytes	Network Output Bytes	Number of bytes sent during the monitoring period Unit: byte	≥ 0	DCS Memcached instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
mc_instantaneous_input_kbps	Input Flow	Instantaneous input traffic Unit: KB/s	≥ 0 KB/s	DCS Memcached instance	1 minute
mc_instantaneous_output_kbps	Output Flow	Instantaneous output traffic Unit: KB/s	≥ 0 KB/s	DCS Memcached instance	1 minute
mc_rejected_connections	Rejected Connections	Number of connections that have exceeded maxclients and been rejected during the monitoring period	≥ 0	DCS Memcached instance	1 minute
mc_expired_keys	Expired Keys	Number of keys that have expired and been deleted during the monitoring period	≥ 0	DCS Memcached instance	1 minute
mc_evicted_keys	Evicted Keys	Number of keys that have been evicted and deleted during the monitoring period	≥ 0	DCS Memcached instance	1 minute
mc_cmd_get	Number of Retrieval Requests	Number of received data retrieval requests	≥ 0	DCS Memcached instance	1 minute
mc_cmd_set	Number of Storage Requests	Number of received data storage requests	≥ 0	DCS Memcached instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
mc_cmd_flush	Number of Flush Requests	Number of received data clearance requests	≥ 0	DCS Memcached instance	1 minute
mc_cmd_touch	Number of Touch Requests	Number of received requests for modifying the validity period of data	≥ 0	DCS Memcached instance	1 minute
mc_get_hits	Number of Retrieval Hits	Number of successful data retrieval operations	≥ 0	DCS Memcached instance	1 minute
mc_get_misses	Number of Retrieval Misses	Number of failed data retrieval operations due to key nonexistence	≥ 0	DCS Memcached instance	1 minute
mc_delete_hits	Number of Delete Hits	Number of successful data deletion operations	≥ 0	DCS Memcached instance	1 minute
mc_delete_misses	Number of Delete Misses	Number of failed data deletion operations due to key nonexistence	≥ 0	DCS Memcached instance	1 minute
mc_incr_hits	Number of Increment Hits	Number of successful increment operations	≥ 0	DCS Memcached instance	1 minute
mc_incr_misses	Number of Increment Misses	Number of failed increment operations due to key nonexistence	≥ 0	DCS Memcached instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
mc_decr_hits	Number of Decrement Hits	Number of successful decrement operations	≥ 0	DCS Memcached instance	1 minute
mc_decr_misses	Number of Decrement Misses	Number of failed decrement operations due to key nonexistence	≥ 0	DCS Memcached instance	1 minute
mc_cas_hits	Number of CAS Hits	Number of successful CAS operations	≥ 0	DCS Memcached instance	1 minute
mc_cas_misses	Number of CAS Misses	Number of failed CAS operations due to key nonexistence	≥ 0	DCS Memcached instance	1 minute
mc_cas_badval	Number of CAS Values Not Matched	Number of failed CAS operations due to CAS value mismatch	≥ 0	DCS Memcached instance	1 minute
mc_touch_hits	Number of Touch Hits	Number of successful requests for modifying the validity period of data	≥ 0	DCS Memcached instance	1 minute
mc_touch_misses	Number of Touch Misses	Number of failed requests for modifying the validity period of data due to key nonexistence	≥ 0	DCS Memcached instance	1 minute
mc_auth_cmds	Authentication Requests	Number of authentication requests	≥ 0	DCS Memcached instance	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
mc_auth_errors	Authentication Failures	Number of failed authentication requests	≥ 0	DCS Memcached instance	1 minute
mc_curr_items	Number of Items Stored	Number of stored data items	≥ 0	DCS Memcached instance	1 minute
mc_command_max_delay	Maximum Command Latency	Maximum latency of commands Unit: ms	≥ 0 ms	DCS Memcached instance	1 minute
mc_is_slow_log_exist	Slow Query Logs	Existence of slow query logs in the instance	● **1**: yes ● **0**: no	DCS Memcached instance	1 minute
mc_keyspace_hits_perc	Hit Rate	Ratio of the number of Memcached cache hits to the number of lookups Unit: %	0–100%	DCS Memcached instance	1 minute

Dimensions

Key	Value
dcs_instance_id	DCS Redis instance
dcs_cluster_redis_node	Redis Server
dcs_cluster_proxy_node	Proxy in a Proxy Cluster DCS Redis 3.0 instance
dcs_memcached_instance_id	DCS Memcached instance

11.2 Common DCS Metrics

This section describes common Redis metrics.

Table 11-7 Common metrics

Metric	Description
CPU Usage	This metric indicates the maximum value in each measurement period (minute-level: every minute; second-level: every 5 seconds). ● For a single-node or master/standby instance, you can view the CPU usage of the instance. ● For a Proxy Cluster instance, you can view the CPU usage of the Redis Servers and the proxies. ● For a Redis Cluster instance, you can only view the CPU usage of the Redis Servers.
Memory Usage	This metric measures the memory usage in each measurement period (minute-level: every minute; second-level: every 5 seconds). ● For a single-node or master/standby instance, you can view the memory usage of the instance. ● For a Proxy Cluster instance, you can view the memory usage of the instance and the proxies. ● For a Redis Cluster instance, you can only view the memory usage of the Redis Servers. **NOTICE** The memory usage does not include the usage of reserved memory.
Connected Clients	This metric indicates the number of instantaneous connected clients, that is, the number of concurrent connections. This metric does not include the number of connections to the standby nodes of master/standby or cluster instances. For details about the maximum allowed number of connections, see the "Max. Connections" column of different instance types listed in **DCS Instance Specifications**.
Ops per Second	This metric indicates the number of operations processed per second. For details about the maximum allowed number of operations per second, see the "Reference Performance (QPS)" column of different instance types listed in **DCS Instance Specifications**.

Metric	Description
Input Flow	This metric indicates the instantaneous input traffic. ● The monitoring data on the instance level shows the aggregated input traffic of all nodes. ● The monitoring data on the node level shows the input traffic of the current node.
Output Flow	This metric indicates the instantaneous output traffic. ● The monitoring data on the instance level shows the aggregated output traffic of all nodes. ● The monitoring data on the node level shows the output traffic of the current node.
Bandwidth Usage	This metric indicates the percentage of the used bandwidth to the maximum bandwidth limit. Bandwidth usage = (Input flow + Output flow)/(2 x Maximum bandwidth) x 100%
Commands Processed	This metric indicates the number of commands processed during the monitoring period. The default monitoring period is 1 minute. The monitoring period of this metric is different from that of the **Ops per Second** metric The **Ops per Second** metric measures the instantaneous number of commands processed. The **Commands Processed** metric measures the total number of commands processed during the monitoring period.
Flow Control Times	This metric indicates the number of times that the maximum allowed bandwidth is exceeded during the monitoring period. For details about the maximum allowed bandwidth, see the "Maximum/Assured Bandwidth" column of different instance types listed in **DCS Instance Specifications**.
Slow Queries	This metric indicates whether slow queries exist on the instance. For details about the cause of a slow query, see **Viewing Redis Slow Queries**.

11.3 Viewing DCS Monitoring Metrics

You can view DCS instance metrics on the **Performance Monitoring** page.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the desired instance.

Step 5 Choose **Performance Monitoring**. All monitoring metrics of the instance are displayed.

NOTE

You can also click **View Metric** in the **Operation** column on the **Cache Manager** page. You will be redirected to the Cloud Eye console. The metrics displayed on the Cloud Eye console are the same as those displayed on the **Performance Monitoring** page of the DCS console.

----End

11.4 Configuring Alarm Rules for Critical Metrics

This section describes the alarm rules of some metrics and how to configure the rules. In actual scenarios, configure alarm rules for metrics by referring to the following alarm policies.

Alarm Policies for DCS Redis Instances

Table 11-8 DCS Redis instance metrics to configure alarm rules for

Metric	Normal Range	Alarm Policy	Approach Upper Limit	Handling Suggestion
CPU Usage	0–100	Alarm threshold: 70 Number of consecutive periods: 2 Alarm severity: Major	No	Consider capacity expansion based on the service analysis. The CPU capacity of a single-node or master/standby instance cannot be expanded. If you need larger capacity, use a cluster instance instead. This metric is available only for Proxy Cluster, single-node, and master/standby instances. For Redis Cluster instances, this metric is available only on the Redis Server level. You can view the metric on the **Redis Server** tab page on the **Performance Monitoring** page of the instance.

Metric	Normal Range	Alarm Policy	Approach Upper Limit	Handling Suggestion
Average CPU Usage	0–100%	Alarm threshold: > 70% Number of consecutive periods: 2 Alarm severity: Major	No	Consider capacity expansion based on the service analysis. The CPU capacity of a single-node or master/standby instance cannot be expanded. If you need larger capacity, use a cluster instance instead. This metric is available only for single-node and master/standby instances. For cluster instances, this metric is available only on the Redis Server level. You can view the metric on the **Redis Server** tab page on the **Performance Monitoring** page of the instance.
Memory Usage	0–100	Alarm threshold: 70 Number of consecutive periods: 2 Alarm severity: Major	No	Expand the capacity of the instance.
Connected Clients	0–10,000	Alarm threshold: 8000 Number of consecutive periods: 2 Alarm severity: Major	No	Optimize the connection pool in the service code to prevent the number of connections from exceeding the maximum limit. For single-node and master/standby instances, the maximum number of connections allowed is 10,000. You can adjust the threshold based on service requirements. Configure this alarm policy on the instance level for single-node and master/standby instances. For cluster instances, configure this alarm policy on the Redis Server and Proxy level.

Metric	Normal Range	Alarm Policy	Approach Upper Limit	Handling Suggestion
New Connections (Count/min)	0–10,000	Alarm threshold: 10,000 Number of consecutive periods: 2 Alarm severity: Minor	-	Check whether **connect** is used and whether the client connection is abnormal. Use persistent connections ("pconnect" in Redis terminology) to ensure performance. Configure this alarm policy on the instance level for single-node and master/standby instances. For cluster instances, configure this alarm policy on the Redis Server and Proxy level.
Input Flow	> 0	Alarm threshold: 80% of the assured bandwidth Number of consecutive periods: 2 Alarm severity: Major	Yes	Consider capacity expansion based on the service analysis and bandwidth limit. Configure this alarm only for single-node and master/standby DCS Redis 3.0 instances and set the alarm threshold to 80% of the assured bandwidth of DCS Redis 3.0 instances.
Output Flow	> 0	Alarm threshold: 80% of the assured bandwidth Number of consecutive periods: 2 Alarm severity: Major	Yes	Consider capacity expansion based on the service analysis and bandwidth limit. Configure this alarm only for single-node and master/standby DCS Redis 3.0 instances and set the alarm threshold to 80% of the assured bandwidth of DCS Redis 3.0 instances.

Alarm Policies for DCS Memcached Instances

Table 11-9 DCS Memcached instance metrics to configure alarm rules for

Metric	Value Range	Alarm Policy	Approach Upper Limit	Handling Suggestion
CPU Usage	0–100%	Alarm threshold: > 70% Number of consecutive periods: 2 Alarm severity: Major	No	Check the service for traffic surge. The CPU capacity of a single-node or master/standby instance cannot be expanded. Analyze the service and consider splitting the service or combine multiple instances into a cluster on the client end.
Memory Usage	0–100%	Alarm threshold: > 65% Number of consecutive periods: 2 Alarm severity: Minor	No	Consider expanding the instance capacity.
Connected Clients	0–10,000	Alarm threshold: > 8000 Number of consecutive periods: 2 Alarm severity: Major	No	Optimize the connection pool in the service code to prevent the number of connections from exceeding the maximum limit.
New Connections	≥ 0	Alarm threshold: > 10,000 Number of consecutive periods: 2 Alarm severity: Minor	-	Check whether **connect** is used and whether the client connection is abnormal. Use persistent connections ("pconnect" in Redis terminology) to ensure performance.

Metric	Value Range	Alarm Policy	Approach Upper Limit	Handling Suggestion
Input Flow	≥ 0	Alarm threshold: > 80% of the assured bandwidth Number of consecutive periods: 2 Alarm severity: Major	Yes	Consider capacity expansion based on the service analysis and bandwidth limit. For details about the bandwidth of different instance specifications, see **Memcached Instance Specifications**.
Output Flow	≥ 0	Alarm threshold: > 80% of the assured bandwidth Number of consecutive periods: 2 Alarm severity: Major	Yes	Consider capacity expansion based on the service analysis and bandwidth limit. For details about the bandwidth of different instance specifications, see **Memcached Instance Specifications**.
Authentication Failures	≥ 0	Alarm threshold: > 0 Number of consecutive periods: 1 Alarm severity: Critical	-	Check whether the password is entered correctly.

Alarm Policies for Redis Server Nodes of Cluster DCS Redis Instances

Table 11-10 Redis server metrics to configure alarm policies for

Metric	Value Range	Alarm Policy	Approach Upper Limit	Handling Suggestion
CPU Usage	0–100%	Alarm threshold: > 70% Number of consecutive periods: 2 Alarm severity: Major	No	Check the service for traffic surge. Check whether the CPU usage is evenly distributed to Redis Server nodes. If the CPU usage is high on multiple nodes, consider capacity expansion. Expanding the capacity of a cluster instance will scale out nodes to share the CPU pressure. If the CPU usage is high on a single node, check whether hot keys exist. If yes, optimize the service code to eliminate hot keys.
Average CPU Usage	0–100%	Alarm threshold: > 70% Number of consecutive periods: 2 Alarm severity: Major	No	Consider capacity expansion based on the service analysis. The CPU capacity of a single-node or master/standby instance cannot be expanded. If you need larger capacity, use a cluster instance instead. This metric is available only for single-node, master/standby, and Proxy Cluster instances. For Redis Cluster instances, this metric is available only on the Redis Server level. You can view the metric on the **Redis Server** tab page on the **Performance Monitoring** page of the instance.

Metric	Value Range	Alarm Policy	Approach Upper Limit	Handling Suggestion
Memory Usage	0–100%	Alarm threshold: > 70% Number of consecutive periods: 2 Alarm severity: Major	No	Check the service for traffic surge. Check whether the memory usage is evenly distributed to Redis Server nodes. If the memory usage is high on multiple nodes, consider capacity expansion. If the memory usage is high on a single node, check whether big keys exist. If yes, optimize the service code to eliminate big keys.
Connected Clients	0–10,000	Alarm threshold: > 8000 Number of consecutive periods: 2 Alarm severity: Major	No	Check whether the number of connections is within the appropriate range. If yes, adjust the alarm threshold.
New Connections	≥ 0	Alarm threshold: > 10,000 Number of consecutive periods: 2 Alarm severity: Minor	-	Check whether **connect** is used. To ensure performance, use persistent connections ("pconnect" in Redis terminology).
Slow Query Logs	0–1	Alarm threshold: > 0 Number of consecutive periods: 1 Alarm severity: Major	-	Use the slow query function on the console to analyze slow commands.

Metric	Value Range	Alarm Policy	Approach Upper Limit	Handling Suggestion
Bandwidth Usage	0–200%	Alarm threshold: > 90% Number of consecutive periods: 2 Alarm severity: Major	Yes	Check whether the bandwidth usage increase comes from read services or write services based on the input and output flow. If the bandwidth usage of a single node is high, check whether big keys exist. Even if the bandwidth usage exceeds 100%, flow control may not necessarily be performed. The actual flow control is subject to the **Flow Control Times** metric. Even if the bandwidth usage is below 100%, flow control may be performed. The real-time bandwidth usage is reported once in every reporting period. The flow control times metric is reported every second. During a reporting period, the traffic may surge within seconds and then fall back. By the time the bandwidth usage is reported, it has restored to the normal level.
Flow Control Times	≥ 0	Alarm threshold: > 0 Number of consecutive periods: 1 Alarm severity: Critical	Yes	Consider capacity expansion based on the specification limits, input flow, and output flow.

Procedure

In the following example, an alarm rule is set for the **CPU Usage** metric.

Step 1 Log in to the DCS console.

Step 2 Click 📍 in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 In the same row as the DCS instance whose metrics you want to view, choose **More** > **View Metric**.

Step 5 Locate the **CPU Usage** metric. Hover over the metric and click + to create an alarm rule for the metric.

The **Create Alarm Rule** page is displayed.

Step 6 Specify the alarm rule details.

1. Specify the alarm policy and alarm severity.
2. Set the alarm notification configurations. If you enable **Alarm Notification**, set the validity period, notification object, and trigger condition.
3. Click **Create**.

NOTE

- For more information about creating alarm rules, see the *Cloud Eye User Guide* > *Using the Alarm Function* > *Creating Alarm Rules*.
- For details about how to modify or disable the created alarms, see "Using the Alarm Function" > "Alarm Rule Management" in the *Cloud Eye User Guide*.

----End

12 Auditing

12.1 Operations That Can Be Recorded by CTS

With CTS, you can query, audit, and review operations performed on cloud resources. Traces include the operation requests sent using the management console or open APIs as well as the results of these requests.

The following lists the DCS operations that can be recorded by CTS.

Table 12-1 DCS operations that can be recorded by CTS

Operation	Resource Type	Trace Name
Creating an instance	Redis	createDCSInstance
Submitting an instance creation request	Redis	submitCreateDCSInstanceRequest
Deleting multiple instances	Redis	batchDeleteDCSInstance
Deleting an instance	Redis	deleteDCSInstance
Modifying instance information	Redis	modifyDCSInstanceInfo
Modifying instance configurations	Redis	modifyDCSInstanceConfig

Operation	Resource Type	Trace Name
Changing instance password	Redis	modifyDCSInstancePassword
Restarting an instance	Redis	restartDCSInstance
Submitting an instance restarting request	Redis	submitRestartDCSInstanceRequest
Starting an instance	Redis	startDCSInstance
Submitting an instance starting request	Redis	submitStartDCSInstanceRequest
Clearing instance data	Redis	flushDCSInstance
Restarting instances in batches	Redis	batchRestartDCSInstance
Submitting a request to restart instances in batches	Redis	submitBatchRestartDCSInstanceRequest
Starting multiple instances	Redis	batchStartDCSInstance
Submitting a request to start instances in batches	Redis	submitBatchStartDCSInstanceRequest
Restoring instance data	Redis	restoreDCSInstance
Submitting a request to restore instance data	Redis	submitRestoreDCSInstanceRequest

Operation	Resource Type	Trace Name
Backing up instance data	Redis	backupDCSInstance
Submitting a request to back up instance data	Redis	submitBackupDCSInstanceRequest
Deleting instance backup files	Redis	deleteInstanceBackupFile
Deleting background tasks	Redis	deleteDCSInstanceJobRecord
Modifying instance specifications	Redis	modifySpecification
Submitting a request to modify instance specifications	Redis	submitModifySpecificationRequest
Creating an instance subscription order	Redis	createInstanceOrder
Updating an enterprise project ID	Redis	updateEnterpriseProjectId
Switching between master and standby nodes	Redis	masterStandbySwitchover
Resetting instance password	Redis	resetDCSInstancePassword

Operation	Resource Type	Trace Name
Submitting a request to clear instance data	Redis	submitFlushDCSInstanceRequest
Accessing Web CLI	Redis	webCliLogin
Running commands in Web CLI	Redis	webCliCommand
Exiting Web CLI	Redis	webCliLogout
Migrating offline data	Redis	offlineMigrate

12.2 Viewing Traces on the CTS Console

After CTS is enabled, the tracker starts recording operations on cloud resources. Operation records for the last seven days can be viewed on the CTS console. This section describes how to query operation records of the last seven days on the CTS console.

Procedure

Step 1 Log in to the management console.

Step 2 Click 📍 in the upper left corner of the management console and select a region and a project.

> NOTE
>
> Select the same region as your application service.

Step 3 Click **Service List** and choose **Management & Deployment** > **Cloud Trace Service**.

Step 4 In the navigation pane, click **Trace List**.

Step 5 Specify the filters used for querying traces. The following filters are available:

- **Trace Type**, **Trace Source**, **Resource Type**, and **Search By**

  Select an option from the drop-down list. Select **DCS** from the **Trace Source** drop-down list.

  When you select **Trace name**, you also need to select a specific trace name.

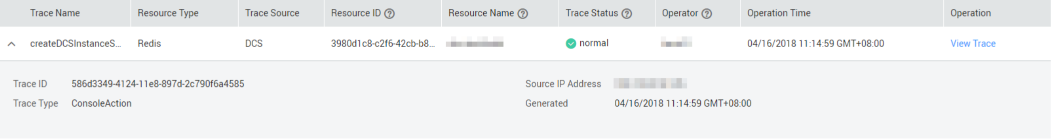
  When you select **Resource ID**, you also need to select a specific resource ID.

When you select **Resource name**, you also need to select a specific resource name.

- **Operator**: Select a specific operator (a user other than tenant).
- **Trace Status**: Available options include **All trace status**, **normal**, **warning**, and **incident**. You can select only one of them.
- Start time and end time: You can specify the time period in which to query traces.

Step 6 Click ⌄ on the left of a trace to expand its details, as shown in **Figure 12-1**.

Figure 12-1 Expanding trace details



	Trace Name	Resource Type	Trace Source	Resource ID ⓘ	Resource Name ⓘ	Trace Status ⓘ	Operator ⓘ	Operation Time	Operation
^	createDCSInstanceS...	Redis	DCS	3980d1c8-c2f6-42cb-b8...	[redacted]	● normal	[redacted]	04/16/2018 11:14:59 GMT+08:00	View Trace

Trace ID 586d3349-4124-11e8-897d-2c790f6a4585

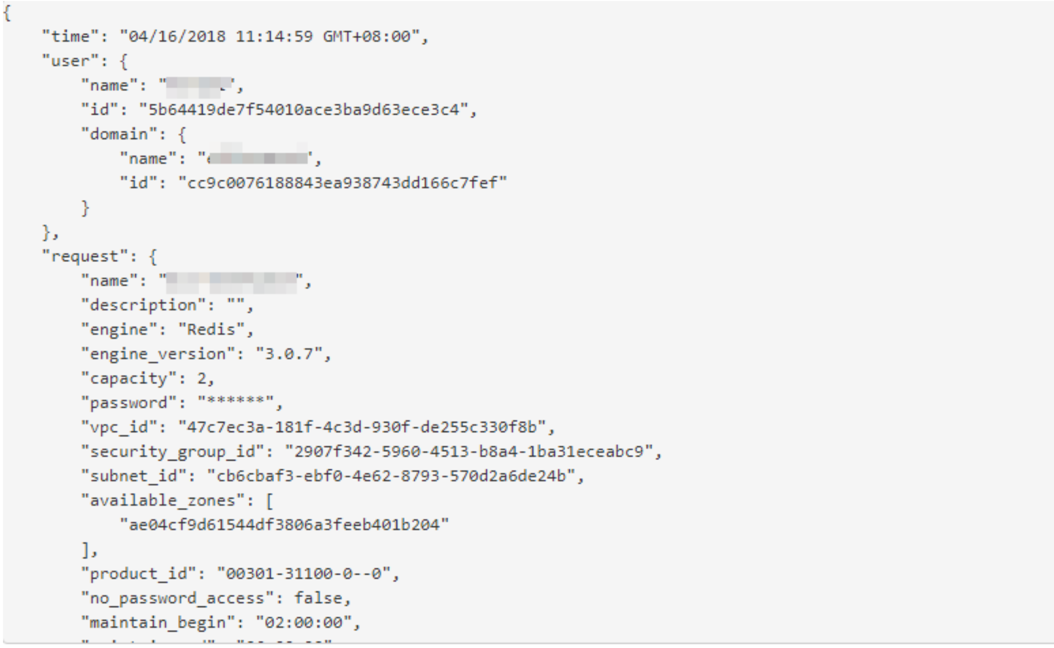
Trace Type ConsoleAction

Source IP Address [redacted]

Generated 04/16/2018 11:14:59 GMT+08:00

Step 7 Click **View Trace** in the **Operation** column. In the dialog box shown in **Figure 12-2**, the trace structure details are displayed.

Figure 12-2 Viewing traces



```
{
    "time": "04/16/2018 11:14:59 GMT+08:00",
    "user": {
        "name": "[redacted]",
        "id": "5b64419de7f54010ace3ba9d63ece3c4",
        "domain": {
            "name": "[redacted]",
            "id": "cc9c0076188843ea938743dd166c7fef"
        }
    },
    "request": {
        "name": "[redacted]",
        "description": "",
        "engine": "Redis",
        "engine_version": "3.0.7",
        "capacity": 2,
        "password": "******",
        "vpc_id": "47c7ec3a-181f-4c3d-930f-de255c330f8b",
        "security_group_id": "2907f342-5960-4513-b8a4-1ba31eceabc9",
        "subnet_id": "cb6cbaf3-ebf0-4e62-8793-570d2a6de24b",
        "available_zones": [
            "ae04cf9d61544df3806a3feeb401b204"
        ],
        "product_id": "00301-31100-0--0",
        "no_password_access": false,
        "maintain_begin": "02:00:00",
```

----End

13 Data Migration Guide

13.1 Overview

This guide provides suggestions and instructions on how to migrate Redis data. For details on how to migrate Memcached data, see **How Do I Migrate Memcached Data?**

Due to variations of Redis application environments and scenarios, migration solutions must be detailed to address actual requirements. The time required for data migration is related to the data volume, the location of source Redis data, and the network bandwidth. Record and evaluate the duration during the rehearsal phase.

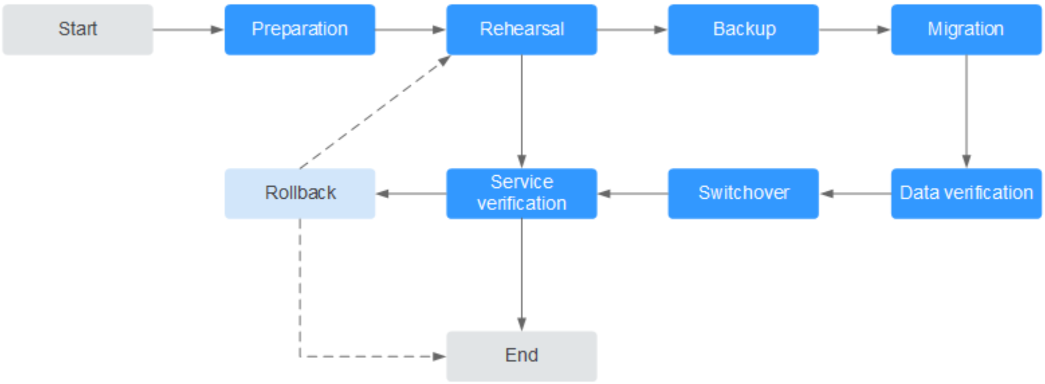
When migrating data, analyze the cache commands (reference: **Command Compatibility**) used by your service systems and verify the commands one by one during the rehearsal phase. If necessary, contact technical support.

NOTICE

- Data migration is an important and stringent task requiring high accuracy and timeliness. It varies depending on specific services and operation environments.
- Cases provided in this document are for reference only. Consider your service scenarios and requirements during actual migration.
- Some commands in this document contain instance passwords, which will be recorded in the operating system (OS). Ensure that the passwords are not disclosed and clear operation records in a timely manner.

13.2 Migration Process

Figure 13-1 Migration flowchart



Evaluation

Collect the following information about the cached data to be migrated (based on **Information to be collected for the migration**):

- Number of instances
- Number of databases (DBs) configured for each instance
- Number of keys in each DB
- DBs used for your services
- Space occupied by each instance
- Redis version
- Redis instance configurations (single-node, master/standby, or cluster)
- Mapping relationships between your services and instances

Plan the following information about DCS instances based on the collected information:

- Number of instances to be applied for
- Specifications and type (single-node, master/standby, or cluster) of each instance
- Virtual Private Clouds (VPCs), security groups, and subnets, and security groups, to which the instances and services belong

NOTE

redis-cli -h ${redis_address} -p ${port}

- Run the following command to query the data distribution and obtain the IDs of DBs with data and the number of keys in each DB:

  info keyspace

  Query and record the number of keys in each DB for subsequent migration verification.

- Run the following command to query the space occupied by the instance data. Check whether the available disk space of Elastic Cloud Servers (ECSs) is sufficient for transition, and whether the instance specifications and remaining available memory are sufficient.

  info memory

  The occupied space can be obtained from the value of **used_memory_human**.

Preparation

After completing the evaluation, prepare the following items:

1. Mobile storage devices

   These devices are used to copy and transfer data in case of network disconnection (in scenarios with data centers of enterprises).

2. Network resources

   Create VPCs and subnets based on service planning.

3. Server resources

   Apply for ECSs to bear Redis clients. The ECSs are used to export or import cached data.

   Recommended ECS specifications are 8 vCPUs | 16 GB or higher.

4. DCS instances

   Create DCS instances based on the migration planning. If the number of instances exceeds the default quota, submit a service ticket or contact technical support.

5. Related tools

   Install the FTP tool and Redis migration tools.

6. Information to be collected

   Collect the contact information of people involved in the migration, server addresses, login credentials, cache instance information, and DB information.

7. Overall migration plan

   Formulate the overall migration plan, including the personnel arrangement, rehearsal, migration, verification, service switchover, and rollback solutions.

   Break down each solution into executable operations and set milestones to mark the end of tasks.

Rehearsal

The rehearsal phase aims to:

1. Verify the feasibility of the migration tools and migration process.
2. Discover problems that may occur during migration and make effective improvements.

3. Evaluate the time required for migration.
4. Optimize the migration steps and verify the feasibility of concurrent implementation of some tasks to improve migration efficiency.

Backup

Before migration, back up related data, including but not limited to cached data and Redis configuration files, in case of emergency.

Migration

After conducting one or two rounds of migration rehearsal and solving problems found in the rehearsal, start data migration.

Break down the migration process into executable steps with specific start and end confirmation actions.

Data Verification

Check the following items:

- The key distribution of each DB is consistent with the original or expected distribution.
- Main keys.
- Expiration time of keys.
- Whether instances can be normally backed up and restored.

Service Switchover

1. After the data migration and verification, use the new instances for your services.
2. If DB IDs are changed, modify the ID configurations for your services.
3. If your services are migrated from data centers or cloud platforms provided by other vendors to this cloud as a whole, services and cached data can be migrated concurrently.

Service Verification

After the service switchover:

1. Verify the connectivity between your service applications and DCS instances.
2. Verify whether cached data can be normally added, deleted, modified, and queried.
3. If possible, perform pressure tests to ensure that the performance satisfies the peak service pressure.

Rollback

If your services are unavailable after the data migration because unexpected problems occur and cannot be solved in the short term, roll back your services.

Since source Redis data still exists, you only need to roll back your services and use the source Redis instances again.

After the rollback, you can continue to restart from the rehearsal or even preparation phase to solve the problems.

Information to be collected for the migration

The following table lists the information to be collected in the evaluation and preparation phases.

Table 13-1 Information to be collected for the migration

Migration Source	Item	Description
Source Redis (List the information about all instances to be migrated.)	Source Redis IP address	-
	Redis instance password (if any)	-
	Total data volume	Obtained from the value of **used_memory_human** by running the **info memory** command. Used to evaluate whether the migration solution, DCS instance specifications, and available disk space of ECSs meet requirements, and to estimate the time required for migration (service interruption duration).
	IDs of DBs with data	Obtained by running the **info keyspace** command. Used to check whether the migration involves multiple DBs and non-AOF files. Some open-source tools can export and import data of only one DB at a time. For DCS instances, the single-node and master/standby types provide 256 DBs (DB 0 to DB 255), and the cluster type provides only one DB by default.
	Number of keys in each DB	Used to verify the data integrity after migration.
	Data type	The Cloud Data Migration (CDM) service supports two data formats: hash and string. If the source data contains data in other formats such as list and set, use a third-party migration tool.

Migration Source	Item	Description
ECS If a large number of instances are to be migrated, prepare multiple ECSs for concurrent migration.	EIP	Select ECSs that can communicate with DCS instances for data import to ensure network stability. Configure high-specification bandwidth to improve data transmission efficiency.
	Login credentials (username and password)	-
	CPU and memory	Some migration tools support concurrent import through multiple threads. High-specification ECSs help improve import efficiency.
	Available disk space	Sufficient available disk space needs to be reserved on the ECSs to store compressed files and decompressed cached data files. Note: To improve data transmission efficiency, compress large-size data files before transmitting them to ECSs.
DCS instances (Select appropriate instance specifications and quantities based on the number of source Redis instances and data volume.)	Instance connection address	-
	Instance connection port	-
	Instance password	-
	Instance type	-
	Instance specifications and available memory	-
Network configurations	VPC	Plan VPCs in advance to ensure that your service applications and DCS instances are in same VPCs.
	Subnet	-

Migration Source	Item	Description
	Whitelist or security group	DCS Redis 3.0, 4.0, 5.0, and 6.0 professional edition instances are deployed in different modes. Therefore, the access control methods vary. You can control access to your DCS instances by setting security groups or whitelists. For details, see **Security Group Configurations** or **Managing IP Address Whitelist**.
...	...	*Other configurations*.

13.3 Migration Solution Notes

Migration Tools

Table 13-2 Comparing Redis migration tools

Tool/ Command/ Service	Feature	Description
DCS console	Supports online migration (in full or incrementally) and backup migration (by importing backup files) with intuitive operations.	• Backup migration is suitable when the source and target Redis instances are not connected, and the source Redis instance does not support the **SYNC** and **PSYNC** commands. To migrate data, import your backup files to OBS, and DCS will read data from OBS and migrate the data to the target DCS Redis instance. • Online migration is suitable when the source Redis instance supports the **SYNC** and **PSYNC** commands. Data in the source Redis instance can be migrated in full or incrementally to the target instance.

Tool/ Command/ Service	Feature	Description
redis-cli	● The Redis command line interface (CLI), which can be used to export data as an RDB file or import the AOF file (that is, all DBs) of an instance. ● An AOF file is large file containing a full set of data change commands.	-
Rump	Supports online migration between DBs of an instance or between DBs of different instances.	Rump does not support incremental migration. Stop services before migrating data. Otherwise, keys might be lost. For details, see **Online Migration from Another Cloud Using Rump**.
redis-shake	An open-source tool that supports both online and offline migration.	redis-shake is suitable for migrating Redis Cluster data.
Self-developed migration script	Flexible and can be adjusted as required.	-

Migration Schemes

NOTE

Self-hosted Redis refers to self-hosted Redis on this service, in another cloud, or in on-premises data centers.

Table 13-3 Migration schemes

Scenario	Tool	Use Case	Description
From self-hosted Redis to DCS	DCS console	● If the network between your self-hosted Redis instance and the DCS Redis instance is connected, follow to the instructions in **Online Migration of Self-Hosted Redis**. ● If the network between your self-hosted Redis instance and the DCS Redis instance is not connected, follow to the instructions in **Backup Migration of Self-Hosted Redis**.	-
	redis-cli	**Self-Hosted Redis Migration with redis-cli (AOF)**	-
		Self-Hosted Redis Migration with redis-cli (RDB)	-
	redis-shake	**Self-Hosted Redis Cluster Migration with redis-shake**	-
Between DCS instances	DCS console	Migrate data from an earlier-version DCS Redis instance to a later-version DCS Redis instance, for example, from a DCS Redis 3.0 instance to a DCS Redis 4.0/5.0/6.0 instance. ● If the network between the source and target DCS Redis instances is connected, follow to the instructions in **Online Migration Between Instances**. ● If the network between the source and target DCS Redis instances is not connected, follow to the instructions in **Backup Import Between Instances**.	**Attempts to migrate data from a later-version Redis instance to an earlier-version Redis instance are not recommended because they will fail** due to data compatibility issues between different Redis versions.

Scenario	Tool	Use Case	Description
		Migrate Redis data between regions. For details, see **Backup Import Between Instances**.	The **SYNC** and **PSYNC** commands are disabled by default for DCS Redis instances. These commands are enabled for online migration within a region, and remain disabled for online migration between regions. Therefore, you can only use backup migration when migrating DCS Redis instance data between regions.
		Migrate Redis data from one account to another. ● For details, see **Backup Import Between Instances**. ● If the DCS Redis instances of the two accounts are connected, you can also follow the instructions in **Online Migration Between Instances**.	-
From another cloud to DCS	DCS console	● If the **SYNC** and **PSYNC** commands are not disabled for the Redis service provided by another cloud, follow the instructions in **Online Migration from Another Cloud**. ● If the **SYNC** and **PSYNC** commands are disabled for the Redis service provided by another cloud, follow the instructions in **Backup Import from Another Cloud**.	If online migration is required, contact the O&M personnel of another cloud to enable the **SYNC** and **PSYNC** commands.

Scenario	Tool	Use Case	Description
	Rump	**Online Migration from Another Cloud Using Rump**	-
	redis-shake	**Backup Import from Another Cloud Using redis-shake**	-
		Online Migration from Another Cloud Using redis-shake	-

13.4 Migrating Data from Self-Hosted Redis to DCS

13.4.1 Online Migration of Self-Hosted Redis

Application Scenarios

If the source and target instances are interconnected and the **SYNC** and **PSYNC** commands are supported by the source instance, data can be migrated online in full or incrementally from the source to the target.

⚠ CAUTION

- If the **SYNC** and **PSYNC** commands are disabled on the source Redis instance, enable them before performing online migration. Otherwise, the migration fails. If you use a DCS Redis instance for online migration, the **SYNC** command is automatically enabled.
- You cannot use public networks for online migration.
- During online migration, you are advised to set **repl-timeout** on the source instance to 300s and **client-output-buffer-limit** to 20% of the maximum memory of the instance.
- The source must be Redis 3.0 or later.

Impacts on Services

During online migration, data is essentially synchronized in full to a new replica. Therefore, perform online migration during low-demand hours.

Prerequisites

- Before migrating data, read through **Migration Tools and Schemes** to learn about the DCS data migration function and select an appropriate target instance.
- By default, a cluster instance has only one DB (DB0). Before you migrate data from a multi-DB single-node or master/standby instance to a Redis Cluster instance, check whether any data exists on databases other than DB0. To

ensure that the migration succeeds, move all data to DB0 by referring to **Online Migration with Rump**.

Step 1: Obtain the Source Redis Address

Obtain the IP address/domain name and port number of the source Redis instance.

Step 2: Prepare the Target DCS Redis Instance

- If a target DCS Redis instance is not available, create one first. For details, see **Creating a DCS Redis Instance**.
- If you already have a DCS Redis instance, you do not need to create one again, but you need to clear the instance data before the migration. For details, see **Clearing DCS Instance Data**.

  If the target instance data is not cleared before the migration and the source and target instances contain the same key, the key in the target instance will be overwritten by the key in the source instance after the migration.

Step 3: Check the Network

Step 1 Check whether the source Redis instance, the target Redis instance, and the migration task are configured with the same VPC.

If yes, go to **Step 4: Create an Online Migration Task**. If no, go to **Step 2**.

Step 2 Check whether the VPCs configured for the source Redis instance, the target Redis instance, and the migration task are connected to ensure that the VM resource of the migration task can access the source and target Redis instances.

If yes, go to **Step 4: Create an Online Migration Task**. If no, go to **Step 3**.

Step 3 Perform the following operations to establish the network.

- If the source and target Redis instances are in the same DCS region, create a VPC peering connection by referring to "VPC Peering Connection" in the *Virtual Private Cloud User Guide*.
- If the source and target Redis instances are on different clouds, create a connection using only Direct Connect. For details, see the *Direct Connect User Guide*.

----End

Step 4: Create an Online Migration Task

Step 1 Log in to the DCS console.

Step 2 In the navigation pane, choose **Data Migration**.

Step 3 Click **Create Online Migration Task**.

Step 4 Enter the task name and description.

Step 5 Configure the VPC, subnet, and security group for the migration task.

The VPC, subnet, and security group facilitate the migration. Ensure that the migration resources can access the source and target Redis instances.

NOTE

- The online migration task uses a tenant IP address (**Migration ECS** displayed on the **Basic Information** page of the task.) If a whitelist is configured for the source or target instance, add the migration IP address to the whitelist or disable the whitelist.
- To allow the VM used by the migration task to access the source and target instances, set an outbound rule for the task's security group to allow traffic through the IP addresses and ports of the source and target instances. By default, all outbound traffic is allowed.

----End

Step 5: Configure the Online Migration Task

Step 1 On the **Online Migration** tab page, click **Configure** in the row containing the online migration task you just created.

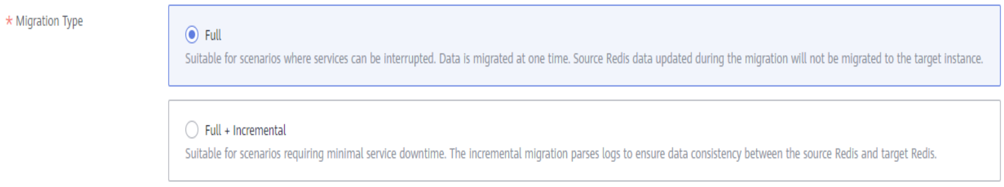
Step 2 Select a migration type.

Supported migration types are **Full** and **Full + Incremental**, which are described in **Table 13-4**.

Table 13-4 Migration type description

Migration Type	Description
Full	Suitable for scenarios where services can be interrupted. Data is migrated at one time. **Source instance data updated during the migration will not be migrated to the target instance.**
Full + incremental	Suitable for scenarios requiring minimal service downtime. The incremental migration parses logs to ensure data consistency between the source and target instances. Once the migration starts, it remains **Migrating** until you click **Stop** in the **Operation** column. After the migration is stopped, data in the source instance will not be lost, but data will not be written to the target instance. When the transmission network is stable, the delay of incremental migration is within seconds. The actual delay depends on the transmission quality of the network link.

Figure 13-2 Selecting the migration type



* Migration Type

● Full
Suitable for scenarios where services can be interrupted. Data is migrated at one time. Source Redis data updated during the migration will not be migrated to the target instance.

○ Full + Incremental
Suitable for scenarios requiring minimal service downtime. The incremental migration parses logs to ensure data consistency between the source Redis and target Redis.

Step 3 Configure source Redis and target Redis.

1. The Redis type can be **Redis in the cloud** or **Self-hosted Redis** as required.
   - **Redis in the cloud**: a DCS Redis instance (source or target) that is in the same VPC as the migration task. If you select this option, specify a DCS Redis instance.
   - **Self-hosted Redis**: a DCS Redis instance, Redis in another cloud, or self-hosted Redis. If you select this option, enter Redis addresses.
2. If the instance is password-protected, click **Test Connection** to check whether the instance password is correct and whether the network is connected. If the instance is not password-protected, click **Test Connection** directly.

Step 4 Click **Next**.

Step 5 Confirm the migration task details and click **Submit**.

Go back to the data migration task list. After the migration is successful, the task status changes to **Successful**.

NOTE

- Once incremental migration starts, it remains **Migrating** until you click **Stop**.
- To stop a migration task, select the check box on the left of the migration task and click **Stop** above the migration task.
- After data migration, duplicate keys will be overwritten.

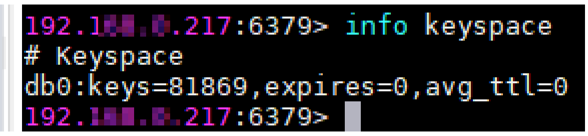
If the migration fails, click the migration task and check the log on the **Migration Logs** page.

----End

Verifying the Migration

After the migration is complete, use redis-cli to connect the source and target Redis instances to check data integrity.

1. Connect to the source Redis and the target Redis.
2. Run the **info keyspace** command to check the values of **keys** and **expires**.



```
192.1[redacted].[redacted].217:6379> info keyspace
# Keyspace
db0:keys=81869,expires=0,avg_ttl=0
192.1[redacted].[redacted].217:6379>
```

3. Calculate the differences between the values of **keys** and **expires** of the source Redis and the target Redis. If the differences are the same, the data is complete and the migration is successful.

During full migration, source Redis data updated during the migration will not be migrated to the target instance.

13.4.2 Backup Migration of Self-Hosted Redis

Application Scenarios

Use the DCS console to migrate Redis data from Redis of another cloud or self-hosted Redis to DCS for Redis.

Simply download the source Redis data and then upload the data to an OBS bucket in the same region as the target DCS Redis instance. After you have created a migration task on the DCS console, DCS will read data from the OBS bucket and data will be migrated to the target instance.

.aof, .rdb, .zip, and .tar.gz files can be uploaded to OBS buckets. You can directly upload .aof and .rdb files or compress them into .zip or .tar.gz files before uploading.

Prerequisites

- The OBS bucket must be in the same region as the target DCS Redis instance.
- The data files to be uploaded must be in the .aof, .rdb, .zip, or .tar.gz format.
- To migrate data from a single-node or master/standby Redis instance of another cloud, create a backup task and download the backup file.
- To migrate data from a cluster Redis instance of another cloud, download all backup files, upload all of them to the OBS bucket, and select all of them for the migration. Each backup file contains data for a shard of the instance.

Step 1: Prepare the Target DCS Redis Instance

- If a target DCS Redis instance is not available, create one first. For details, see **Creating a DCS Redis Instance**.
- If you already have a DCS Redis instance, you do not need to create one again, but you need to clear the instance data before the migration. For details, see **Clearing DCS Instance Data**.

Step 2: Create an OBS Bucket and Upload Backup Files

Step 1 Create an OBS bucket.

1. Log in to the OBS Console and click **Create Bucket**.
2. Select a region.

   The OBS bucket must be in the same region as the target DCS Redis instance.
3. Specify **Bucket Name**.

   The bucket name must meet the naming rules specified on the console.
4. Set **Storage Class** to **Standard**, **Warm** or **Cold**.
5. Set **Bucket Policy** to **Private**, **Public Read**, or **Public Read and Write**.
6. Configure default encryption.
7. Click **Create Now**.

Step 2 Upload the backup data files to the OBS bucket by using OBS Browser+.

If the backup file to be uploaded does not exceed 5 GB, upload the file using the OBS console by referring to step **Step 3**.

If the backup file to be uploaded is larger than 5 GB, perform the following steps to upload the file using OBS Browser+.

1. Download OBS Browser+.

   For details, see section "Downloading OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)*.

2. Install OBS Browser+.

   For details, see section "Installing OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)*.

3. Log in to OBS Browser+.

   For details, see section "Logging In to OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)*.

4. Create a bucket.

   For details, see the *OBS Console Operation Guide* > "Getting Started" > "Creating a Bucket".

5. Upload backup data.

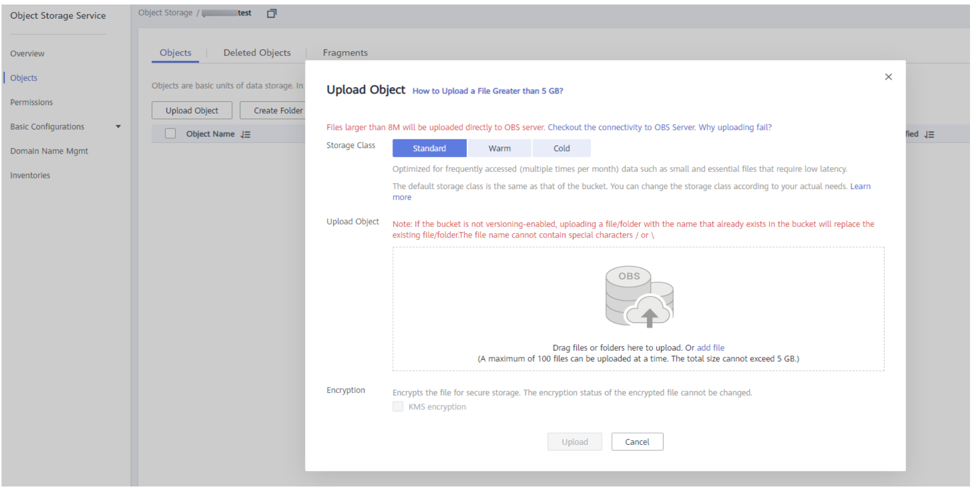
Step 3 On the OBS console, upload the backup data files to the OBS bucket.

Perform the following steps if the backup file size does not exceed 5 GB:

1. In the bucket list, click the name of the created bucket.
2. In the navigation pane, choose **Objects**.
3. On the **Objects** tab page, click **Upload Object**.
4. Upload the objects.

   To upload objects, drag files or folders to the **Upload Object** area or click **add file**. A maximum of 100 files can be uploaded at a time. The total size cannot exceed 5 GB.

Figure 13-3 Uploading an object



Objects are basic units of data storage. In

Files larger than 8M will be uploaded directly to OBS server. Checkout the connectivity to OBS Server. Why uploading fail?

Storage Class: Standard | Warm | Cold

Optimized for frequently accessed (multiple times per month) data such as small and essential files that require low latency.

The default storage class is the same as that of the bucket. You can change the storage class according to your actual needs. Learn more

Upload Object: Note: If the bucket is not versioning-enabled, uploading a file/folder with the name that already exists in the bucket will replace the existing file/folder.The file name cannot contain special characters / or \

Drag files or folders here to upload. Or add file
(A maximum of 100 files can be uploaded at a time. The total size cannot exceed 5 GB.)

Encryption: Encrypts the file for secure storage. The encryption status of the encrypted file cannot be changed.

□ KMS encryption

5. (Optional) Select **KMS encryption** to encrypt the file you want to upload.
6. Click **Upload**.

----End

Step 3: Create a Migration Task

Step 1 Log in to the DCS console.

Step 2 In the navigation pane, choose **Data Migration**.

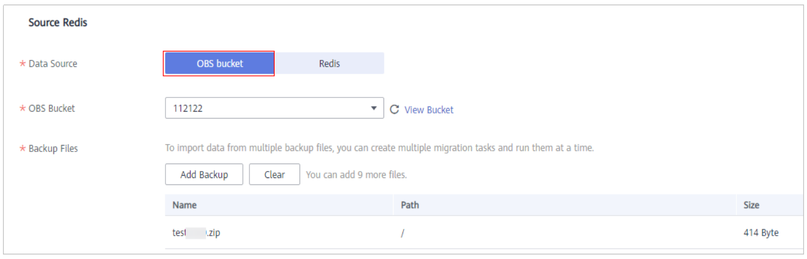
Step 3 Click **Create Backup Import Task**.

Step 4 Enter the task name and description.

Step 5 In the **Source Redis** area, select **OBS Bucket** for **Data Source** and then select the OBS bucket to which you have uploaded backup files.

Step 6 Click **Add Backup** and select the backup files to be migrated.

Figure 13-4 Specifying the backup file information



Source Redis

* Data Source: OBS bucket | Redis

* OBS Bucket: 112122 ▾ View Bucket

* Backup Files: To import data from multiple backup files, you can create multiple migration tasks and run them at a time.

Add Backup | Clear | You can add 9 more files.

Name	Path	Size
test[redacted].zip	/	414 Byte

Step 7 In the **Target Redis** area, select the **Target Redis Instance** prepared in **Step 1: Prepare the Target DCS Redis Instance**.

Step 8 If the target Redis instance has a password, enter the password and click **Test Connection** to check whether the password is correct. If the instance is not password-protected, click **Test Connection** directly.

Step 9 Click **Next**.

Step 10 Confirm the migration task details and click **Submit**.

Go back to the data migration task list. After the migration is successful, the task status changes to **Successful**.

----End

13.4.3 Self-Hosted Redis Migration with redis-cli (AOF)

Introduction

redis-cli is the command line tool of Redis, which can be used after you install the Redis server.

Run the following command to download Redis:

wget http://download.redis.io/releases/redis-5.0.8.tar.gz

This section describes how to use redis-cli to migrate a data from a self-hosted Redis instance to a DCS instance.

Step 1: Generating an AOF File

NOTICE

- Before data migration, suspend your services so that data changes newly generated will not be lost during the migration.
- Migrate data during off-peak hours.

Run the following command to enable cache persistence and obtain an AOF persistence file:

```
redis-cli -h {source_redis_address} -p 6379 -a {password} config set appendonly yes
```

If the size of the AOF file does not change after you have enabled persistence, the AOF file contains full cached data.

NOTE

- To find out the path for storing the AOF file, use redis-cli to access the Redis instance, and run the **config get dir** command. Unless otherwise specified, the file is named as **appendonly.aof** by default.
- To disable synchronization after the AOF file is generated, use redis-cli to log in to the Redis instance and run the **config set appendonly no** command.

Step 2: Uploading the AOF file to ECS

1. To save the transmission time, compress the AOF file before transmission.
2. Upload the compressed file to ECS using an appropriate mode (for example, SFTP mode).

NOTE

Ensure that the ECS has sufficient disk space for data file decompression, and can communicate with the DCS instance. Generally, the ECS and DCS instance are configured to belong to the same VPC and subnet, and the configured security group rules do not restrict access ports. For details about how to configure a security group, see **Security Group Configurations**.

Step 3: Importing Data

```
redis-cli -h {dcs_instance_address} -p 6379 -a {password} --pipe < appendonly.aof
```

Step 4: Verifying Migration

After the data is imported successfully, access the DCS instance and run the **info** command to check whether the data has been successfully imported as required.

If the data import fails, analyze the cause, modify the data import statement, run the **flushall** or **flushdb** command to clear the cached data in the instance, and import the data again.

Efficiency of Data Export and Import

An AOF file can be generated quickly. It applies to scenarios where you can access the Redis server and modify the configurations, such as scenarios with self-built Redis servers.

It takes 4s to 10s to import 1 million data records (20 bytes per data record) in a VPC.

13.4.4 Self-Hosted Redis Migration with redis-cli (RDB)

Introduction

redis-cli is the command line tool of Redis, which can be used after you install the Redis server.

redis-cli supports data export as an RDB file. If your Redis service does not support AOF file export, use redis-cli to obtain an RDB file. Then, use another tool (such as redis-shake) to import the file to a DCS instance.

Operations described in this section are performed on the Linux OS.

Run the following command to download Redis. redis-cli can be used after installation and compilation.

wget http://download.redis.io/releases/redis-5.0.8.tar.gz

NOTICE

The source Redis instance must support the **SYNC** command, which is required when exporting the RDB file using redis-cli.

The **SYNC** command is not supported by DCS Reds 4.0/5.0/6.0 instances and cannot be used to export RDB files. To back up master/standby instance data, use the backup and restoration function provided by the DCS console.

Step 1: Preparation for Data Export

For master/standby or cluster DCS instances, there is a delay in writing data into an RDB file based on the delay policies configured in the **redis.conf** file. Therefore, before data export, learn the RDB policy configurations of the Redis instance to be migrated, suspend your service systems, and then write the required number of test keys into the Redis instance. This ensures that the RDB file is newly generated.

For the Redis service provided by a third-party cloud platform, you can contact its technical support to learn data writing policy configurations of an RDB file.

For example, the default RDB policy configurations in the **redis.conf** file are as follows:

```
save 900 1 //Writes changed data into an RDB file if there is any data change within 900s.
save 300 10 //Writes changed data into an RDB file if there are more than 10 data changes within 300s.
save 60 10000 //Writes changed data into an RDB file if there are more than 10,000 data changes within
60s.
```

Based on the preceding policy configurations, after stopping your service systems from writing data into the Redis instances, you can manually write test data to trigger the policies, so that all service data can be synchronized to the RDB file.

You can delete the test data after data import.

NOTE

If there is any DB not used by your service systems, you can write test data into the DB, and run the **flushdb** command to clear the DB after importing data into DCS.

Step 2: Exporting an RDB File

NOTICE

1. Migrate data during off-peak hours.
2. When exporting Redis Cluster data, individually export the data of each node in the cluster, and then import the data node by node.

Run the following command to export the RDB file:

redis-cli -h ***{source_redis_address}*** **-p** 6379 **-a** ***{password}*** **--rdb** ***{output.rdb}***

If "Transfer finished with success." is displayed after the command is executed, the file is exported successfully.

Step 3: Uploading the RDB File to ECS

1. To save the transmission time, compress the RDB file before transmission.
2. Upload the compressed file to ECS using an appropriate mode (for example, SFTP mode).

NOTE

Ensure that the ECS has sufficient disk space for data file decompression, and can communicate with the DCS instance. Generally, the ECS and DCS instance are configured to belong to the same VPC and subnet, and the configured security group rules do not restrict access ports. For details about how to configure a security group, see **Security Group Configurations**.

Step 4: Importing Data

Use redis-shake to import data.

Step 5: Verifying Migration

After the data is imported successfully, access the DCS instance and run the **info** command to check whether the data has been successfully imported as required.

If the data import fails, analyze the cause, modify the data import statement, run the **flushall** or **flushdb** command to clear the cached data in the instance, and import the data again.

Efficiency of Data Export and Import

Compared with master/standby instances, single-node instances without data persistence configured require a longer time for export of an RDB file, because the RDB file is temporarily generated.

It takes 4s to 10s to import 1 million data records (20 bytes per data record) in a VPC.

13.4.5 Self-Hosted Redis Cluster Migration with redis-shake

redis-shake is an open-source tool for migrating data online or offline (by importing backup files) between Redis Clusters. Data can be migrated to DCS

Redis Cluster instances seamlessly because DCS Redis Cluster inherits the native Redis Cluster design.

The following describes how to use redis-shake to migrate data to a DCS Redis Cluster instance.

Migrating Data Online

You can migrate data online from a self-hosted Redis Cluster to a DCS Redis Cluster instance as long as the two clusters are directly connected or connected through a transit server.

Data in Redis Clusters of another cloud cannot be migrated online because the **SYNC** and **PSYNC** commands are disabled by some vendors.

1. Create a Redis Cluster instance on the DCS console.

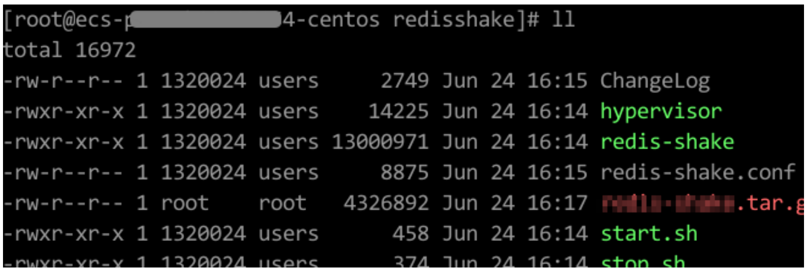
   The memory of this instance cannot be smaller than that of the source Redis.

2. Prepare a cloud server and install redis-shake.

   redis-shake must be able to access both the source and target Redis. Bound an EIP to the cloud server.

   You can use ECS and configure the same VPC, subnet, and security group for the ECS and the DCS instance. If the source Redis is deployed on cloud servers of another cloud, allow public access to the servers.

   Download and decompress the release version of redis-shake. (The following uses v2.1.2 as an example. You can also use **other redis-shake versions**.)



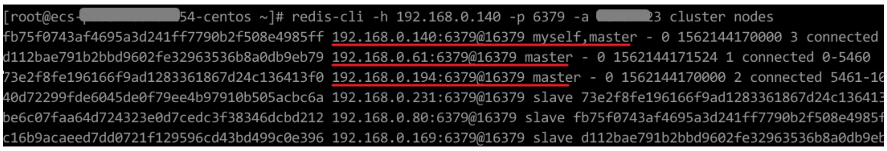
```
[root@ecs-[redacted]4-centos redisshake]# ll
total 16972
-rw-r--r-- 1 1320024 users     2749 Jun 24 16:15 ChangeLog
-rwxr-xr-x 1 1320024 users    14225 Jun 24 16:14 hypervisor
-rwxr-xr-x 1 1320024 users 13000971 Jun 24 16:14 redis-shake
-rw-r--r-- 1 1320024 users     8875 Jun 24 16:15 redis-shake.conf
-rw-r--r-- 1 root    root   4326892 Jun 24 16:17 [redacted].tar.g
-rwxr-xr-x 1 1320024 users      458 Jun 24 16:14 start.sh
-rwxr-xr-x 1 1320024 users      374 Jun 24 16:14 stop.sh
```

3. Locate the masters of the source and target Redis Clusters and obtain the IP addresses of the masters.

   Online data migration must be performed node by node. Run the following command to query the IP addresses and port numbers of all nodes in both the source and target Redis Clusters.

   redis-cli -h *{redis_address}* -p *{redis_port}* -a *{redis_password}* cluster nodes

   In the command output similar to the following, obtain the IP addresses and ports of all masters.



```
[root@ecs-[redacted]54-centos ~]# redis-cli -h 192.168.0.140 -p 6379 -a [redacted]3 cluster nodes
fb75f0743af4695a3d241ff7790b2f508e4985ff 192.168.0.140:6379@16379 myself,master - 0 1562144170000 3 connected
d112bae791b2bbd9602fe32963536b8a0db9eb79 192.168.0.61:6379@16379 master - 0 1562144171524 1 connected 0-5460
73e2f8fe196166f9ad1283361867d24c136413f0 192.168.0.194:6379@16379 master - 0 1562144170000 2 connected 5461-1
40d72299fde6045de0f79ee4b97910b505acbc6a 192.168.0.231:6379@16379 slave 73e2f8fe196166f9ad1283361867d24c13641
be6c07faa64d724323e0d7cedc3f38346dcbd212 192.168.0.80:6379@16379 slave fb75f0743af4695a3d241ff7790b2f508e4985
c16b9acaeed7dd0721f129596cd43bd499c0e396 192.168.0.169:6379@16379 slave d112bae791b2bbd9602fe32963536b8a0db9e
```

NOTE

After Redis is installed, it runs with redis-cli. To install Redis on CentOS, run the **yum install redis** command.

4. Edit the redis-shake configuration file.

   Edit the **redis-shake.conf** file by providing the following information about all the masters of both the source and the target:

   ```
   source.type = cluster
   # If there is no password, skip the following parameter.
   source.password_raw = {source_redis_password}
   # IP addresses and port numbers of all masters of the source Redis Cluster, which are separated by semicolons (;).
   source.address = {master1_ip}:{master1_port};{master2_ip}:{master2_port}…{masterN_ip}:{masterN_port}
   target.type = cluster
   # If there is no password, skip the following parameter.
   target.password_raw = {target_redis_password}
   # IP addresses and port numbers of all masters of the target instance, which are separated by semicolons (;).
   target.address = {master1_ip}:{master1_port};{master2_ip}:{master2_port}…{masterN_ip}:{masterN_port}
   ```

   Save and exit.

5. Migrate data online.

   Run the following command to synchronize data between the source and the target Redis:

   ./redis-shake -type sync -conf redis-shake.conf

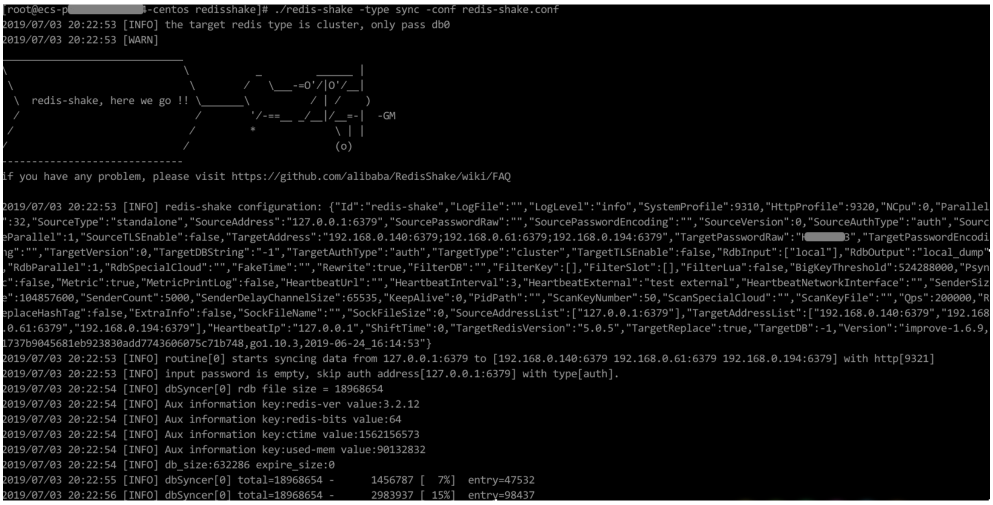
   If the following information is displayed, the full synchronization has been completed and incremental synchronization begins.

   ```
   sync rdb done.
   ```

   If the following information is displayed, no new data is incremented. You can stop the incremental synchronization by pressing **Ctrl**+**C**.

   ```
   sync:  +forwardCommands=0  +filterCommands=0  +writeBytes=0
   ```

   Figure 13-5 Online migration using redis-shake



```
[root@ecs-p[redacted]4-centos redisshake]# ./redis-shake -type sync -conf redis-shake.conf
2019/07/03 20:22:53 [INFO] the target redis type is cluster, only pass db0
2019/07/03 20:22:53 [WARN]
______________________________
\                             \           _         ______ |
 \                             \        /   \___-=O'/|O'/__|
  \  redis-shake, here we go !! \_______\          / | /    )
  /                             /        '/-==__ _/__|/__=-|  -GM
 /                             /         *             \ | |
/                             /                        (o)
------------------------------
if you have any problem, please visit https://github.com/alibaba/RedisShake/wiki/FAQ

2019/07/03 20:22:53 [INFO] redis-shake configuration: {"Id":"redis-shake","LogFile":"","LogLevel":"info","SystemProfile":9310,"HttpProfile":9320,"NCpu":0,"Parallel
":32,"SourceType":"standalone","SourceAddress":"127.0.0.1:6379","SourcePasswordRaw":"","SourcePasswordEncoding":"","SourceVersion":0,"SourceAuthType":"auth","Sourc
eParallel":1,"SourceTLSEnable":false,"TargetAddress":"192.168.0.140:6379;192.168.0.61:6379;192.168.0.194:6379","TargetPasswordRaw":"[redacted]","TargetPasswordEncodi
ng":"","TargetVersion":0,"TargetDBString":"-1","TargetAuthType":"auth","TargetType":"cluster","TargetTLSEnable":false,"RdbInput":["local"],"RdbOutput":"local_dump"
,"RdbParallel":1,"RdbSpecialCloud":"","FakeTime":"","Rewrite":true,"FilterDB":"","FilterKey":[],"FilterSlot":[],"FilterLua":false,"BigKeyThreshold":524288000,"Psyn
c":false,"Metric":true,"MetricPrintLog":false,"HeartbeatUrl":"","HeartbeatInterval":3,"HeartbeatExternal":"test external","HeartbeatNetworkInterface":"","SenderSiz
e":104857600,"SenderCount":5000,"SenderDelayChannelSize":65535,"KeepAlive":0,"PidPath":"","ScanKeyNumber":50,"ScanSpecialCloud":"","ScanKeyFile":"","Qps":200000,"R
eplaceHashTag":false,"ExtraInfo":false,"SockFileName":"","SockFileSize":0,"SourceAddressList":["127.0.0.1:6379"],"TargetAddressList":["192.168.0.140:6379","192.168
.0.61:6379","192.168.0.194:6379"],"HeartbeatIp":"127.0.0.1","ShiftTime":0,"TargetRedisVersion":"5.0.5","TargetReplace":true,"TargetDB":-1,"Version":"improve-1.6.9,
1737b9045681eb923830add7743606075c71b748,go1.10.3,2019-06-24_16:14:53"}
2019/07/03 20:22:53 [INFO] routine[0] starts syncing data from 127.0.0.1:6379 to [192.168.0.140:6379 192.168.0.61:6379 192.168.0.194:6379] with http[9321]
2019/07/03 20:22:53 [INFO] input password is empty, skip auth address[127.0.0.1:6379] with type[auth].
2019/07/03 20:22:54 [INFO] dbSyncer[0] rdb file size = 18968654
2019/07/03 20:22:54 [INFO] Aux information key:redis-ver value:3.2.12
2019/07/03 20:22:54 [INFO] Aux information key:redis-bits value:64
2019/07/03 20:22:54 [INFO] Aux information key:ctime value:1562156573
2019/07/03 20:22:54 [INFO] Aux information key:used-mem value:90132832
2019/07/03 20:22:54 [INFO] db_size:632286 expire_size:0
2019/07/03 20:22:55 [INFO] dbSyncer[0] total=18968654 -      1456787 [  7%]  entry=47532
2019/07/03 20:22:56 [INFO] dbSyncer[0] total=18968654 -      2983937 [ 15%]  entry=98437
```

6. Verify the migration.

   After data synchronization, access the DCS Redis Cluster instance using redis-cli. Run the **info** command to query the number of keys in the **Keyspace** section to confirm that data has been fully imported.

   If the data has not been fully imported, run the **flushall** or **flushdb** command to clear the cached data in the instance, and synchronize data again.

7. Clear the redis-shake configuration file.

Importing Backup Files

If the source Redis and the destitution Redis cannot be connected, or the source Redis is deployed on other clouds, you can migrate data by importing backup files.

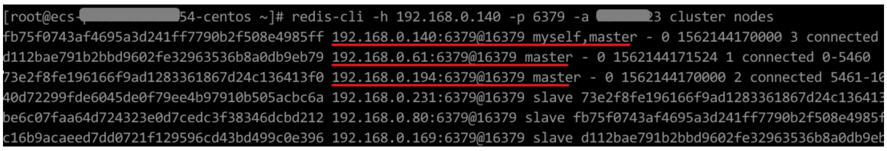
1. Create a Redis Cluster instance on the DCS console.

   The memory of this instance cannot be smaller than that of the source Redis.

2. Run the following command to obtain the IP addresses and port numbers of all masters of the source Redis and target Redis:

   redis-cli -h *{redis_address}* -p *{redis_port}* -a *{redis_password}* cluster nodes

   In the command output similar to the following, obtain the IP addresses and ports of all masters.

```
[root@ecs-[redacted]54-centos ~]# redis-cli -h 192.168.0.140 -p 6379 -a [redacted]23 cluster nodes
fb75f0743af4695a3d241ff7790b2f508e4985ff 192.168.0.140:6379@16379 myself,master - 0 1562144170000 3 connected
d112bae791b2bbd9602fe32963536b8a0db9eb79 192.168.0.61:6379@16379 master - 0 1562144171524 1 connected 0-5460
73e2f8fe196166f9ad1283361867d24c136413f0 192.168.0.194:6379@16379 master - 0 1562144170000 2 connected 5461-10
40d72299fde6045de0f79ee4b97910b505acbc6a 192.168.0.231:6379@16379 slave 73e2f8fe196166f9ad1283361867d24c13641
be6c07faa64d724323e0d7cedc3f38346dcbd212 192.168.0.80:6379@16379 slave fb75f0743af4695a3d241ff7790b2f508e4985f
c16b9acaeed7dd0721f129596cd43bd499c0e396 192.168.0.169:6379@16379 slave d112bae791b2bbd9602fe32963536b8a0db9eb
```

   📖 NOTE

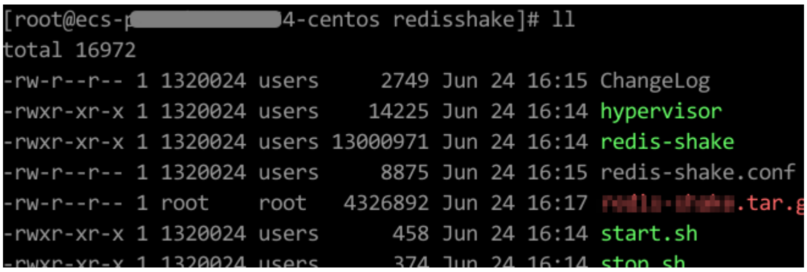
   After Redis is installed, it runs with redis-cli. To install Redis on CentOS, run the **yum install redis** command.

3. Prepare a cloud server and install redis-shake.

   redis-shake must be able to access the target Redis and bound to an EIP.

   You can use ECS and configure the same VPC, subnet, and security group for the ECS and the DCS instance.

   Download and decompress the release version of redis-shake. (The following uses v2.1.2 as an example.)

```
[root@ecs-[redacted]4-centos redisshake]# ll
total 16972
-rw-r--r-- 1 1320024 users     2749 Jun 24 16:15 ChangeLog
-rwxr-xr-x 1 1320024 users    14225 Jun 24 16:14 hypervisor
-rwxr-xr-x 1 1320024 users 13000971 Jun 24 16:14 redis-shake
-rw-r--r-- 1 1320024 users     8875 Jun 24 16:15 redis-shake.conf
-rw-r--r-- 1 root    root   4326892 Jun 24 16:17 [redacted].tar.g
-rwxr-xr-x 1 1320024 users      458 Jun 24 16:14 start.sh
-rwxr-xr-x 1 1320024 users      374 Jun 24 16:14 stop.sh
```

   📖 NOTE

   If the source Redis is deployed in the data center intranet, install redis-shake on the intranet server. Export data and then upload the data to the cloud server as instructed by the following steps

4. Export the RDB file.

   – Edit the **redis-shake.conf** file by providing the following information about all the masters of both the source and the target:

```
source.type = cluster
# If there is no password, skip the following parameter.
source.password_raw = {source_redis_password}
# IP addresses and port numbers of all masters of the source Redis Cluster, which are separated by semicolons (;).
source.address = {master1_ip}:{master1_port};{master2_ip}:{master2_port}…{masterN_ip}:{masterN_port}
```

- Run the following command to export the RDB file:

  ./redis-shake -type dump -conf redis-shake.conf

  If the following information is displayed in the execution log, the backup file is exported successfully:

  ```
  execute runner[*run.CmdDump] finished!
  ```

5. Import the RDB file.
   a. Import the RDB file (or files) to the cloud server. The cloud server must be connected to the target DCS instance.
   b. Edit the redis-shake configuration file.

      Edit the **redis-shake.conf** file by providing the following information about all the masters of both the source and the target:

      ```
      target.type = cluster
      # If there is no password, skip the following parameter.
      target.password_raw = {target_redis_password}
      # IP addresses and port numbers of all masters of the target instance, which are separated by
      semicolons (;).
      target.address = {master1_ip}:{master1_port};{master2_ip}:{master2_port}…{masterN_ip}:
      {masterN_port}
      # List the RDB files to be imported, separated by semicolons (;).
      rdb.input = {local_dump.0};{local_dump.1};{local_dump.2};{local_dump.3}
      ```

      Save and exit.
   c. Run the following command to import the RDB file to the target instance:

      ./redis-shake -type restore -conf redis-shake.conf

      If the following information is displayed in the execution log, the backup file is imported successfully:

      ```
      Enabled http stats, set status (incr), and wait forever.
      ```

6. Verify the migration.

   After data synchronization, access the DCS Redis Cluster instance using redis-cli. Run the **info** command to query the number of keys in the **Keyspace** section to confirm that data has been fully imported.

   If the data has not been fully imported, run the **flushall** or **flushdb** command to clear the cached data in the instance, and synchronize data again.

13.5 Migrating Data Between DCS Instances

13.5.1 Online Migration Between Instances

Application Scenarios

If the source and target instances are interconnected and the **SYNC** and **PSYNC** commands are supported by the source instance, data can be migrated online in full or incrementally from the source to the target.

⚠ CAUTION

- If the **SYNC** and **PSYNC** commands are disabled on the source Redis instance, enable them before performing online migration. Otherwise, the migration fails. If you use a DCS Redis instance for online migration, the **SYNC** command is automatically enabled.
- You cannot use public networks for online migration.
- During online migration, you are advised to set **repl-timeout** on the source instance to 300s and **client-output-buffer-limit** to 20% of the maximum memory of the instance.
- The source must be Redis 3.0 or later.

Impacts on Services

During online migration, data is essentially synchronized in full to a new replica. Therefore, perform online migration during low-demand hours.

Prerequisites

- Before migrating data, read through **Migration Tools and Schemes** to learn about the DCS data migration function and select an appropriate target instance.
- By default, a cluster instance has only one DB (DB0). Before you migrate data from a multi-DB single-node or master/standby instance to a Redis Cluster instance, check whether any data exists on databases other than DB0. To ensure that the migration succeeds, move all data to DB0 by referring to **Online Migration with Rump**.

Step 1: Obtain the Source Redis Address

Obtain the IP address/domain name and port number of the source Redis instance.

Step 2: Prepare the Target DCS Redis Instance

- If a target DCS Redis instance is not available, create one first. For details, see **Creating a DCS Redis Instance**.
- If you already have a DCS Redis instance, you do not need to create one again, but you need to clear the instance data before the migration. For details, see **Clearing DCS Instance Data**.

  If the target instance data is not cleared before the migration and the source and target instances contain the same key, the key in the target instance will be overwritten by the key in the source instance after the migration.

Step 3: Check the Network

Step 1 Check whether the source Redis instance, the target Redis instance, and the migration task are configured with the same VPC.

If yes, go to **Step 4: Create an Online Migration Task**. If no, go to **Step 2**.

Step 2 Check whether the VPCs configured for the source Redis instance, the target Redis instance, and the migration task are connected to ensure that the VM resource of the migration task can access the source and target Redis instances.

If yes, go to **Step 4: Create an Online Migration Task**. If no, go to **Step 3**.

Step 3 Perform the following operations to establish the network.

- If the source and target Redis instances are in the same DCS region, create a VPC peering connection by referring to "VPC Peering Connection" in the *Virtual Private Cloud User Guide*.
- If the source and target Redis instances are on different clouds, create a connection using only Direct Connect. For details, see the *Direct Connect User Guide*.

----End

Step 4: Create an Online Migration Task

Step 1 Log in to the DCS console.

Step 2 In the navigation pane, choose **Data Migration**.

Step 3 Click **Create Online Migration Task**.

Step 4 Enter the task name and description.

Step 5 Configure the VPC, subnet, and security group for the migration task.

The VPC, subnet, and security group facilitate the migration. Ensure that the migration resources can access the source and target Redis instances.

NOTE

- The online migration task uses a tenant IP address (**Migration ECS** displayed on the **Basic Information** page of the task.) If a whitelist is configured for the source or target instance, add the migration IP address to the whitelist or disable the whitelist.
- To allow the VM used by the migration task to access the source and target instances, set an outbound rule for the task's security group to allow traffic through the IP addresses and ports of the source and target instances. By default, all outbound traffic is allowed.

----End

Step 5: Configure the Online Migration Task

Step 1 On the **Online Migration** tab page, click **Configure** in the row containing the online migration task you just created.

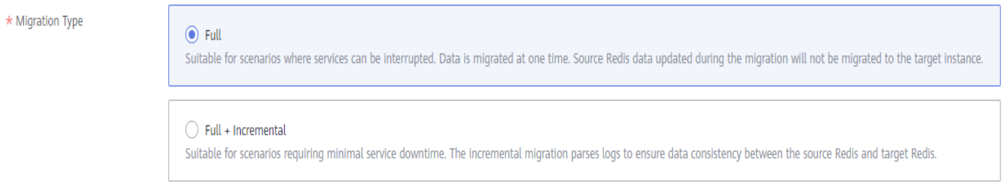
Step 2 Select a migration type.

Supported migration types are **Full** and **Full + Incremental**, which are described in **Table 13-5**.

Table 13-5 Migration type description

Migration Type	Description
Full	Suitable for scenarios where services can be interrupted. Data is migrated at one time. **Source instance data updated during the migration will not be migrated to the target instance.**
Full + incremental	Suitable for scenarios requiring minimal service downtime. The incremental migration parses logs to ensure data consistency between the source and target instances. Once the migration starts, it remains **Migrating** until you click **Stop** in the **Operation** column. After the migration is stopped, data in the source instance will not be lost, but data will not be written to the target instance. When the transmission network is stable, the delay of incremental migration is within seconds. The actual delay depends on the transmission quality of the network link.

Figure 13-6 Selecting the migration type



* Migration Type

● Full
Suitable for scenarios where services can be interrupted. Data is migrated at one time. Source Redis data updated during the migration will not be migrated to the target instance.

○ Full + Incremental
Suitable for scenarios requiring minimal service downtime. The incremental migration parses logs to ensure data consistency between the source Redis and target Redis.

Step 3 Configure source Redis and target Redis.

1. The Redis type can be **Redis in the cloud** or **Self-hosted Redis** as required.
   - **Redis in the cloud**: a DCS Redis instance (source or target) that is in the same VPC as the migration task. If you select this option, specify a DCS Redis instance.
   - **Self-hosted Redis**: a DCS Redis instance, Redis in another cloud, or self-hosted Redis. If you select this option, enter Redis addresses.
2. If the instance is password-protected, click **Test Connection** to check whether the instance password is correct and whether the network is connected. If the instance is not password-protected, click **Test Connection** directly.

Step 4 Click **Next**.

Step 5 Confirm the migration task details and click **Submit**.

Go back to the data migration task list. After the migration is successful, the task status changes to **Successful**.

NOTE

- Once incremental migration starts, it remains **Migrating** until you click **Stop**.
- To stop a migration task, select the check box on the left of the migration task and click **Stop** above the migration task.
- After data migration, duplicate keys will be overwritten.

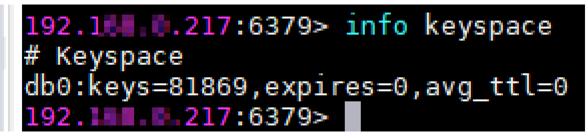
If the migration fails, click the migration task and check the log on the **Migration Logs** page.

----End

Verifying the Migration

After the migration is complete, use redis-cli to connect the source and target Redis instances to check data integrity.

1. Connect to the source Redis and the target Redis.
2. Run the **info keyspace** command to check the values of **keys** and **expires**.



```
192.1[redacted].[redacted].217:6379> info keyspace
# Keyspace
db0:keys=81869,expires=0,avg_ttl=0
192.[redacted].[redacted].217:6379>
```

3. Calculate the differences between the values of **keys** and **expires** of the source Redis and the target Redis. If the differences are the same, the data is complete and the migration is successful.

During full migration, source Redis data updated during the migration will not be migrated to the target instance.

13.5.2 Backup Import Between Instances

Application Scenarios

Use the DCS console to migrate Redis data from Redis of another cloud or self-hosted Redis to DCS for Redis.

Simply download the source Redis data and then upload the data to an OBS bucket in the same region as the target DCS Redis instance. After you have created a migration task on the DCS console, DCS will read data from the OBS bucket and data will be migrated to the target instance.

.aof, .rdb, .zip, and .tar.gz files can be uploaded to OBS buckets. You can directly upload .aof and .rdb files or compress them into .zip or .tar.gz files before uploading.

Prerequisites

- The OBS bucket must be in the same region as the target DCS Redis instance.
- The data files to be uploaded must be in the .aof, .rdb, .zip, or .tar.gz format.
- To migrate data from a single-node or master/standby Redis instance of another cloud, create a backup task and download the backup file.

- To migrate data from a cluster Redis instance of another cloud, download all backup files, upload all of them to the OBS bucket, and select all of them for the migration. Each backup file contains data for a shard of the instance.

Step 1: Prepare the Target DCS Redis Instance

- If a target DCS Redis instance is not available, create one first. For details, see **Creating a DCS Redis Instance**.
- If you already have a DCS Redis instance, you do not need to create one again, but you need to clear the instance data before the migration. For details, see **Clearing DCS Instance Data**.

Step 2: Create an OBS Bucket and Upload Backup Files

Step 1 Create an OBS bucket.

1. Log in to the OBS Console and click **Create Bucket**.
2. Select a region.

   The OBS bucket must be in the same region as the target DCS Redis instance.
3. Specify **Bucket Name**.

   The bucket name must meet the naming rules specified on the console.
4. Set **Storage Class** to **Standard**, **Warm** or **Cold**.
5. Set **Bucket Policy** to **Private**, **Public Read**, or **Public Read and Write**.
6. Configure default encryption.
7. Click **Create Now**.

Step 2 Upload the backup data files to the OBS bucket by using OBS Browser+.

If the backup file to be uploaded does not exceed 5 GB, upload the file using the OBS console by referring to step **Step 3**.

If the backup file to be uploaded is larger than 5 GB, perform the following steps to upload the file using OBS Browser+.

1. Download OBS Browser+.

   For details, see section "Downloading OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)*.
2. Install OBS Browser+.

   For details, see section "Installing OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)*.
3. Log in to OBS Browser+.

   For details, see section "Logging In to OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)*.
4. Create a bucket.

   For details, see the *OBS Console Operation Guide* > "Getting Started" > "Creating a Bucket".
5. Upload backup data.

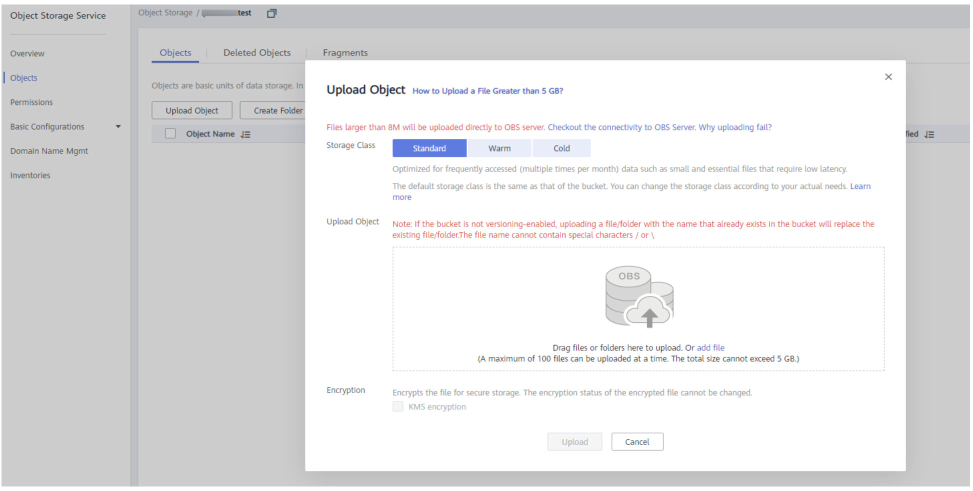
Step 3 On the OBS console, upload the backup data files to the OBS bucket.

Perform the following steps if the backup file size does not exceed 5 GB:

1. In the bucket list, click the name of the created bucket.
2. In the navigation pane, choose **Objects**.
3. On the **Objects** tab page, click **Upload Object**.
4. Upload the objects.

   To upload objects, drag files or folders to the **Upload Object** area or click **add file**. A maximum of 100 files can be uploaded at a time. The total size cannot exceed 5 GB.

   Figure 13-7 Uploading an object

   Objects are basic units of data storage. In

   Upload Object How to Upload a File Greater than 5 GB?

   Files larger than 8M will be uploaded directly to OBS server. Checkout the connectivity to OBS Server. Why uploading fail?

   Storage Class: Standard | Warm | Cold

   Optimized for frequently accessed (multiple times per month) data such as small and essential files that require low latency.

   The default storage class is the same as that of the bucket. You can change the storage class according to your actual needs. Learn more

   Upload Object: Note: If the bucket is not versioning-enabled, uploading a file/folder with the name that already exists in the bucket will replace the existing file/folder.The file name cannot contain special characters / or \

   Drag files or folders here to upload. Or add file
   (A maximum of 100 files can be uploaded at a time. The total size cannot exceed 5 GB.)

   Encryption: Encrypts the file for secure storage. The encryption status of the encrypted file cannot be changed.

   □ KMS encryption

5. (Optional) Select **KMS encryption** to encrypt the file you want to upload.
6. Click **Upload**.

----End

Step 3: Create a Migration Task

Step 1 Log in to the DCS console.

Step 2 In the navigation pane, choose **Data Migration**.

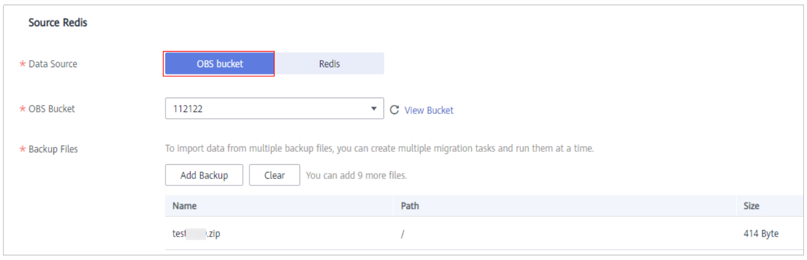
Step 3 Click **Create Backup Import Task**.

Step 4 Enter the task name and description.

Step 5 In the **Source Redis** area, select **OBS Bucket** for **Data Source** and then select the OBS bucket to which you have uploaded backup files.

Step 6 Click **Add Backup** and select the backup files to be migrated.

Figure 13-8 Specifying the backup file information



Source Redis

* Data Source: OBS bucket | Redis

* OBS Bucket: 112122

* Backup Files: To import data from multiple backup files, you can create multiple migration tasks and run them at a time.

You can add 9 more files.

Name	Path	Size
tes[redacted].zip	/	414 Byte

Step 7 In the **Target Redis** area, select the **Target Redis Instance** prepared in **Step 1: Prepare the Target DCS Redis Instance**.

Step 8 If the target Redis instance has a password, enter the password and click **Test Connection** to check whether the password is correct. If the instance is not password-protected, click **Test Connection** directly.

Step 9 Click **Next**.

Step 10 Confirm the migration task details and click **Submit**.

Go back to the data migration task list. After the migration is successful, the task status changes to **Successful**.

----End

13.6 Migration from Another Cloud

13.6.1 Online Migration from Another Cloud

Application Scenarios

If the source and target instances are interconnected and the **SYNC** and **PSYNC** commands are supported by the source instance, data can be migrated online in full or incrementally from the source to the target.

⚠ CAUTION

- If the **SYNC** and **PSYNC** commands are disabled on the source Redis instance, enable them before performing online migration. Otherwise, the migration fails. If you use a DCS Redis instance for online migration, the **SYNC** command is automatically enabled.
- You cannot use public networks for online migration.
- During online migration, you are advised to set **repl-timeout** on the source instance to 300s and **client-output-buffer-limit** to 20% of the maximum memory of the instance.
- The source must be Redis 3.0 or later.

Impacts on Services

During online migration, data is essentially synchronized in full to a new replica. Therefore, perform online migration during low-demand hours.

Prerequisites

- Before migrating data, read through **Migration Tools and Schemes** to learn about the DCS data migration function and select an appropriate target instance.
- By default, a cluster instance has only one DB (DB0). Before you migrate data from a multi-DB single-node or master/standby instance to a Redis Cluster

instance, check whether any data exists on databases other than DB0. To ensure that the migration succeeds, move all data to DB0 by referring to **Online Migration with Rump**.

Step 1: Obtain the Source Redis Address

Obtain the IP address/domain name and port number of the source Redis instance.

Step 2: Prepare the Target DCS Redis Instance

- If a target DCS Redis instance is not available, create one first. For details, see **Creating a DCS Redis Instance**.
- If you already have a DCS Redis instance, you do not need to create one again, but you need to clear the instance data before the migration. For details, see **Clearing DCS Instance Data**.

  If the target instance data is not cleared before the migration and the source and target instances contain the same key, the key in the target instance will be overwritten by the key in the source instance after the migration.

Step 3: Check the Network

Step 1 Check whether the source Redis instance, the target Redis instance, and the migration task are configured with the same VPC.

If yes, go to **Step 4: Create an Online Migration Task**. If no, go to **Step 2**.

Step 2 Check whether the VPCs configured for the source Redis instance, the target Redis instance, and the migration task are connected to ensure that the VM resource of the migration task can access the source and target Redis instances.

If yes, go to **Step 4: Create an Online Migration Task**. If no, go to **Step 3**.

Step 3 Perform the following operations to establish the network.

- If the source and target Redis instances are in the same DCS region, create a VPC peering connection by referring to "VPC Peering Connection" in the *Virtual Private Cloud User Guide*.
- If the source and target Redis instances are on different clouds, create a connection using only Direct Connect. For details, see the *Direct Connect User Guide*.

----End

Step 4: Create an Online Migration Task

Step 1 Log in to the DCS console.

Step 2 In the navigation pane, choose **Data Migration**.

Step 3 Click **Create Online Migration Task**.

Step 4 Enter the task name and description.

Step 5 Configure the VPC, subnet, and security group for the migration task.

The VPC, subnet, and security group facilitate the migration. Ensure that the migration resources can access the source and target Redis instances.

NOTE

- The online migration task uses a tenant IP address (**Migration ECS** displayed on the **Basic Information** page of the task.) If a whitelist is configured for the source or target instance, add the migration IP address to the whitelist or disable the whitelist.
- To allow the VM used by the migration task to access the source and target instances, set an outbound rule for the task's security group to allow traffic through the IP addresses and ports of the source and target instances. By default, all outbound traffic is allowed.

----End

Step 5: Configure the Online Migration Task

Step 1 On the **Online Migration** tab page, click **Configure** in the row containing the online migration task you just created.

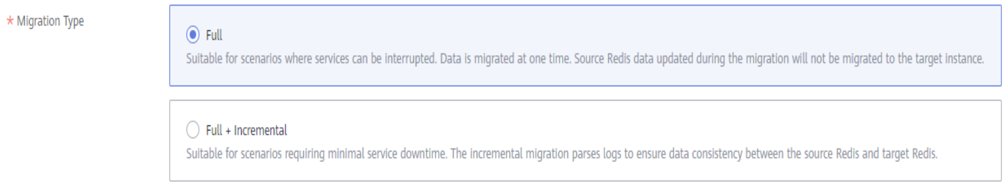
Step 2 Select a migration type.

Supported migration types are **Full** and **Full + Incremental**, which are described in **Table 13-6**.

Table 13-6 Migration type description

Migration Type	Description
Full	Suitable for scenarios where services can be interrupted. Data is migrated at one time. **Source instance data updated during the migration will not be migrated to the target instance.**
Full + incremental	Suitable for scenarios requiring minimal service downtime. The incremental migration parses logs to ensure data consistency between the source and target instances. Once the migration starts, it remains **Migrating** until you click **Stop** in the **Operation** column. After the migration is stopped, data in the source instance will not be lost, but data will not be written to the target instance. When the transmission network is stable, the delay of incremental migration is within seconds. The actual delay depends on the transmission quality of the network link.

Figure 13-9 Selecting the migration type



* Migration Type

● Full
Suitable for scenarios where services can be interrupted. Data is migrated at one time. Source Redis data updated during the migration will not be migrated to the target instance.

○ Full + Incremental
Suitable for scenarios requiring minimal service downtime. The incremental migration parses logs to ensure data consistency between the source Redis and target Redis.

Step 3 Configure source Redis and target Redis.

1. The Redis type can be **Redis in the cloud** or **Self-hosted Redis** as required.
   - **Redis in the cloud**: a DCS Redis instance (source or target) that is in the same VPC as the migration task. If you select this option, specify a DCS Redis instance.
   - **Self-hosted Redis**: a DCS Redis instance, Redis in another cloud, or self-hosted Redis. If you select this option, enter Redis addresses.
2. If the instance is password-protected, click **Test Connection** to check whether the instance password is correct and whether the network is connected. If the instance is not password-protected, click **Test Connection** directly.

Step 4 Click **Next**.

Step 5 Confirm the migration task details and click **Submit**.

Go back to the data migration task list. After the migration is successful, the task status changes to **Successful**.

NOTE

- Once incremental migration starts, it remains **Migrating** until you click **Stop**.
- To stop a migration task, select the check box on the left of the migration task and click **Stop** above the migration task.
- After data migration, duplicate keys will be overwritten.

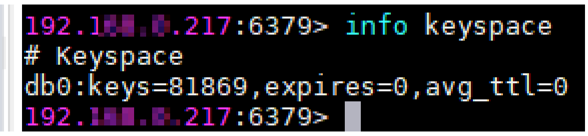
If the migration fails, click the migration task and check the log on the **Migration Logs** page.

----End

Verifying the Migration

After the migration is complete, use redis-cli to connect the source and target Redis instances to check data integrity.

1. Connect to the source Redis and the target Redis.
2. Run the **info keyspace** command to check the values of **keys** and **expires**.



```
192.1[redacted].[redacted].217:6379> info keyspace
# Keyspace
db0:keys=81869,expires=0,avg_ttl=0
192.1[redacted].[redacted].217:6379>
```

3. Calculate the differences between the values of **keys** and **expires** of the source Redis and the target Redis. If the differences are the same, the data is complete and the migration is successful.

During full migration, source Redis data updated during the migration will not be migrated to the target instance.

13.6.2 Backup Import from Another Cloud

Application Scenarios

Use the DCS console to migrate Redis data from Redis of another cloud or self-hosted Redis to DCS for Redis.

Simply download the source Redis data and then upload the data to an OBS bucket in the same region as the target DCS Redis instance. After you have created a migration task on the DCS console, DCS will read data from the OBS bucket and data will be migrated to the target instance.

.aof, .rdb, .zip, and .tar.gz files can be uploaded to OBS buckets. You can directly upload .aof and .rdb files or compress them into .zip or .tar.gz files before uploading.

Prerequisites

- The OBS bucket must be in the same region as the target DCS Redis instance.
- The data files to be uploaded must be in the .aof, .rdb, .zip, or .tar.gz format.
- To migrate data from a single-node or master/standby Redis instance of another cloud, create a backup task and download the backup file.
- To migrate data from a cluster Redis instance of another cloud, download all backup files, upload all of them to the OBS bucket, and select all of them for the migration. Each backup file contains data for a shard of the instance.

Step 1: Prepare the Target DCS Redis Instance

- If a target DCS Redis instance is not available, create one first. For details, see **Creating a DCS Redis Instance**.
- If you already have a DCS Redis instance, you do not need to create one again, but you need to clear the instance data before the migration. For details, see **Clearing DCS Instance Data**.

Step 2: Create an OBS Bucket and Upload Backup Files

Step 1 Create an OBS bucket.

1. Log in to the OBS Console and click **Create Bucket**.
2. Select a region.

   The OBS bucket must be in the same region as the target DCS Redis instance.
3. Specify **Bucket Name**.

   The bucket name must meet the naming rules specified on the console.
4. Set **Storage Class** to **Standard**, **Warm** or **Cold**.
5. Set **Bucket Policy** to **Private**, **Public Read**, or **Public Read and Write**.
6. Configure default encryption.
7. Click **Create Now**.

Step 2 Upload the backup data files to the OBS bucket by using OBS Browser+.

If the backup file to be uploaded does not exceed 5 GB, upload the file using the OBS console by referring to step **Step 3**.

If the backup file to be uploaded is larger than 5 GB, perform the following steps to upload the file using OBS Browser+.

1. Download OBS Browser+.

   For details, see section "Downloading OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)*.

2. Install OBS Browser+.

   For details, see section "Installing OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)*.

3. Log in to OBS Browser+.

   For details, see section "Logging In to OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)*.

4. Create a bucket.

   For details, see the *OBS Console Operation Guide* > "Getting Started" > "Creating a Bucket".

5. Upload backup data.

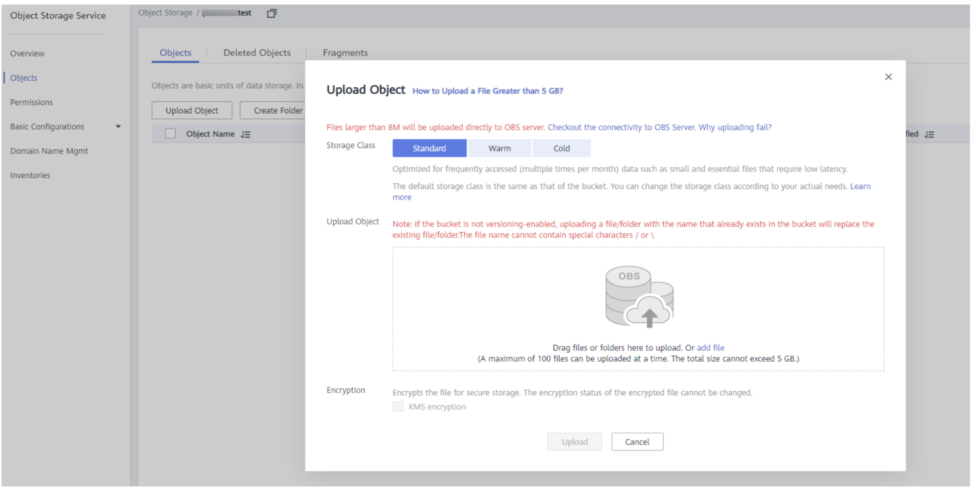
Step 3 On the OBS console, upload the backup data files to the OBS bucket.

Perform the following steps if the backup file size does not exceed 5 GB:

1. In the bucket list, click the name of the created bucket.
2. In the navigation pane, choose **Objects**.
3. On the **Objects** tab page, click **Upload Object**.
4. Upload the objects.

   To upload objects, drag files or folders to the **Upload Object** area or click **add file**. A maximum of 100 files can be uploaded at a time. The total size cannot exceed 5 GB.

Figure 13-10 Uploading an object



Object Storage Service

Overview

Objects

Permissions

Basic Configurations

Domain Name Mgmt

Inventories

Object Storage / [redacted]test

Objects | Deleted Objects | Fragments

Objects are basic units of data storage. In

Upload Object   Create Folder

□ Object Name

Upload Object How to Upload a File Greater than 5 GB?

Files larger than 8M will be uploaded directly to OBS server. Checkout the connectivity to OBS Server. Why uploading fail?

Storage Class: Standard | Warm | Cold

Optimized for frequently accessed (multiple times per month) data such as small and essential files that require low latency.

The default storage class is the same as that of the bucket. You can change the storage class according to your actual needs. Learn more

Upload Object: Note: If the bucket is not versioning-enabled, uploading a file/folder with the name that already exists in the bucket will replace the existing file/folder.The file name cannot contain special characters / or \

Drag files or folders here to upload. Or add file
(A maximum of 100 files can be uploaded at a time. The total size cannot exceed 5 GB.)

Encryption: Encrypts the file for secure storage. The encryption status of the encrypted file cannot be changed.

□ KMS encryption

Upload   Cancel

5. (Optional) Select **KMS encryption** to encrypt the file you want to upload.
6. Click **Upload**.

----End

Step 3: Create a Migration Task

Step 1 Log in to the DCS console.

Step 2 In the navigation pane, choose **Data Migration**.

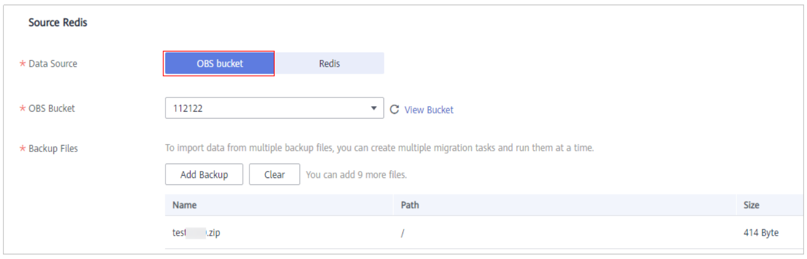
Step 3 Click **Create Backup Import Task**.

Step 4 Enter the task name and description.

Step 5 In the **Source Redis** area, select **OBS Bucket** for **Data Source** and then select the OBS bucket to which you have uploaded backup files.

Step 6 Click **Add Backup** and select the backup files to be migrated.

Figure 13-11 Specifying the backup file information



Source Redis

* Data Source: OBS bucket | Redis

* OBS Bucket: 112122 ▾ View Bucket

* Backup Files: To import data from multiple backup files, you can create multiple migration tasks and run them at a time.

Add Backup | Clear | You can add 9 more files.

Name	Path	Size
test[redacted].zip	/	414 Byte

Step 7 In the **Target Redis** area, select the **Target Redis Instance** prepared in **Step 1: Prepare the Target DCS Redis Instance**.

Step 8 If the target Redis instance has a password, enter the password and click **Test Connection** to check whether the password is correct. If the instance is not password-protected, click **Test Connection** directly.

Step 9 Click **Next**.

Step 10 Confirm the migration task details and click **Submit**.

Go back to the data migration task list. After the migration is successful, the task status changes to **Successful**.

----End

13.6.3 Online Migration from Another Cloud Using Rump

Background

- Redis instances provided by some cloud service vendors do not allow **SLAVEOF**, **BGSAVE**, and **PSYNC** commands to be issued from Redis clients. As a result, redis-cli, redis-shake, and other tools cannot be used to export data.
- Using the **KEYS** command may block Redis.
- Cloud service vendors usually only support downloading backup files. This method is suitable only for offline migration, featuring longer service interruption.

Rump is an open-source tool designed for migrating Redis data online. It supports migration between DBs of the same instance and between DBs of different instances.

Migration Principles

Rump uses the **SCAN** command to acquire keys and the **DUMP**/**RESTORE** command to get or set values.

Featuring time complexity O(1), **SCAN** is capable of quickly getting all keys. **DUMP**/**RESTORE** is used to read/write values independent from the key type.

Rump brings the following benefits:

- The **SCAN** command replaces the **KEYS** command to avoid blocking Redis.
- Any type of data can be migrated.
- **SCAN** and **DUMP**/**RESTORE** operations are pipelined, improving the network efficiency during data migration.
- No temporary file is involved, saving disk space.
- Buffered channels are used to optimize performance of the source server.

NOTICE

1. To cluster DCS instances, you cannot use Rump. Instead, use redis-shake or redis-cli.
2. To prevent migration command resolution errors, do not include special characters (#@:) in the instance password.
3. Stop the service before migrating data. If data is kept being written in during the migration, some keys might be lost.

Step 1: Installing Rump

1. Download **Rump (release version)**.

   On 64-bit Linux, run the following command:

   wget https://github.com/stickermule/rump/releases/download/0.0.3/rump-0.0.3-linux-amd64;

2. After decompression, run the following commands to add the execution permission:

   mv rump-0.0.3-linux-amd64 rump;

   chmod +x rump;

Step 2: Migrating Data

rump -from {*source_redis_address*} **-to** {*target_redis_address*}

Parameter/Option description:

- *{source_redis_address}*

  Source Redis instance address, in the format of redis://[user:password@]host:port/db. **[user:password@]** is optional. If the instance is accessed in password-protected mode, you must specify the password in the RFC 3986 format. **user** can be omitted, but the colon (:) cannot be omitted. For example, the address may be **redis://:mypassword@192.168.0.45:6379/1**.

  db is the sequence number of the database. If it is not specified, the default value is 0.

- {*target_redis_address*}

  Address of the target Redis instance, in the same format as the source.

  In the following example, data in DB0 of the source Redis is migrated to the target Redis whose connection address is 192.168.0.153. ****** stands for the password.

```
[root@ecs ~]# ./rump -from redis://127.0.0.1:6379/0  -to redis://:******@192.168.0.153:6379/0
.Sync done.
[root@ecs ~]#
```

13.6.4 Backup Import from Another Cloud Using redis-shake

redis-shake is an open-source tool for migrating data online or offline (by importing backup files) between Redis Clusters. If the source Redis Cluster is deployed in another cloud, and online migration is not supported, you can migrate data by importing backup files.

The following describes how to use redis-shake for backup migration to a DCS Redis Cluster instance.

Importing Backup Files

If the source Redis and the destitution Redis cannot be connected, or the source Redis is deployed on other clouds, you can migrate data by importing backup files.

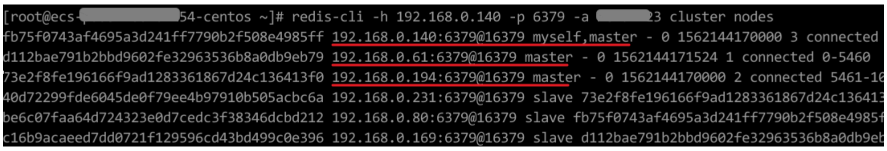
1. Create a Redis Cluster instance on the DCS console.

   The memory of this instance cannot be smaller than that of the source Redis.

2. Run the following command to obtain the IP addresses and port numbers of all masters of the source Redis and target Redis:

   redis-cli -h *{redis_address}* -p *{redis_port}* -a *{redis_password}* cluster nodes

   In the command output similar to the following, obtain the IP addresses and ports of all masters.



```
[root@ecs-[redacted]54-centos ~]# redis-cli -h 192.168.0.140 -p 6379 -a [redacted]23 cluster nodes
fb75f0743af4695a3d241ff7790b2f508e4985ff 192.168.0.140:6379@16379 myself,master - 0 1562144170000 3 connected
d112bae791b2bbd9602fe32963536b8a0db9eb79 192.168.0.61:6379@16379 master - 0 1562144171524 1 connected 0-5460
73e2f8fe196166f9ad1283361867d24c136413f0 192.168.0.194:6379@16379 master - 0 1562144170000 2 connected 5461-1€
40d72299fde6045de0f79ee4b97910b505acbc6a 192.168.0.231:6379@16379 slave 73e2f8fe196166f9ad1283361867d24c136413
be6c07faa64d724323e0d7cedc3f38346dcbd212 192.168.0.80:6379@16379 slave fb75f0743af4695a3d241ff7790b2f508e4985f
c16b9acaeed7dd0721f129596cd43bd499c0e396 192.168.0.169:6379@16379 slave d112bae791b2bbd9602fe32963536b8a0db9eb
```

   NOTE

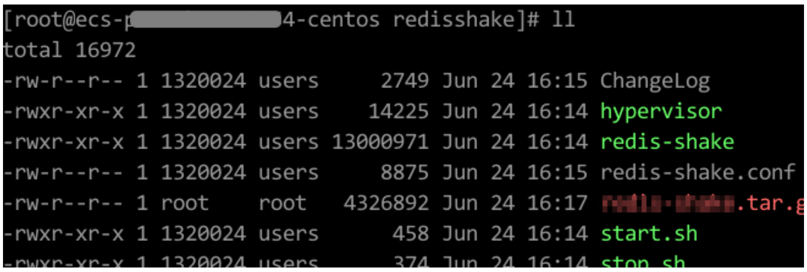
   After Redis is installed, it runs with redis-cli. To install Redis on CentOS, run the **yum install redis** command.

3. Prepare a cloud server and install redis-shake.

   redis-shake must be able to access the target Redis and bound to an EIP.

   You can use ECS and configure the same VPC, subnet, and security group for the ECS and the DCS instance.

   Download and decompress the release version of redis-shake. (The following uses v2.1.2 as an example. You can also use **other redis-shake versions**.)



```
[root@ecs-[redacted]4-centos redisshake]# ll
total 16972
-rw-r--r-- 1 1320024 users     2749 Jun 24 16:15 ChangeLog
-rwxr-xr-x 1 1320024 users    14225 Jun 24 16:14 hypervisor
-rwxr-xr-x 1 1320024 users 13000971 Jun 24 16:14 redis-shake
-rw-r--r-- 1 1320024 users     8875 Jun 24 16:15 redis-shake.conf
-rw-r--r-- 1 root    root   4326892 Jun 24 16:17 [redacted].tar.g
-rwxr-xr-x 1 1320024 users      458 Jun 24 16:14 start.sh
-rwxr-xr-x 1 1320024 users      374 Jun 24 16:14 stop.sh
```

NOTE

If the source Redis is deployed in the data center intranet, install redis-shake on the intranet server. Export data and then upload the data to the cloud server as instructed by the following steps

4. Export the RDB file.
   - Edit the **redis-shake.conf** file by providing the following information about all the masters of both the source and the target:

```
source.type = cluster
# If there is no password, skip the following parameter.
source.password_raw = {source_redis_password}
# IP addresses and port numbers of all masters of the source Redis Cluster, which are separated by semicolons (;).
source.address = {master1_ip}:{master1_port};{master2_ip}:{master2_port}…{masterN_ip}:{masterN_port}
```

   - Run the following command to export the RDB file:

     ./redis-shake -type dump -conf redis-shake.conf

     If the following information is displayed in the execution log, the backup file is exported successfully:

```
execute runner[*run.CmdDump] finished!
```

NOTE

If you cannot export the source Redis backup files using this method due to cloud vendors' restrictions on the **SYNC** and **PSYNC** commands, export the files on the source console or contact the source technical support.

5. Import the RDB file.
   a. Import the RDB file (or files) to the cloud server. The cloud server must be connected to the target DCS instance.
   b. Edit the redis-shake configuration file.

      Edit the **redis-shake.conf** file by providing the following information about all the masters of both the source and the target:

```
target.type = cluster
# If there is no password, skip the following parameter.
target.password_raw = {target_redis_password}
# IP addresses and port numbers of all masters of the target instance, which are separated by semicolons (;).
target.address = {master1_ip}:{master1_port};{master2_ip}:{master2_port}…{masterN_ip}:{masterN_port}
# List the RDB files to be imported, separated by semicolons (;).
rdb.input = {local_dump.0};{local_dump.1};{local_dump.2};{local_dump.3}
```

      Save and exit.

   c. Run the following command to import the RDB file to the target instance:

      ./redis-shake -type restore -conf redis-shake.conf

      If the following information is displayed in the execution log, the backup file is imported successfully:

```
Enabled http stats, set status (incr), and wait forever.
```

6. Verify the migration.

   After data synchronization, access the DCS Redis Cluster instance using redis-cli. Run the **info** command to query the number of keys in the **Keyspace** section to confirm that data has been fully imported.

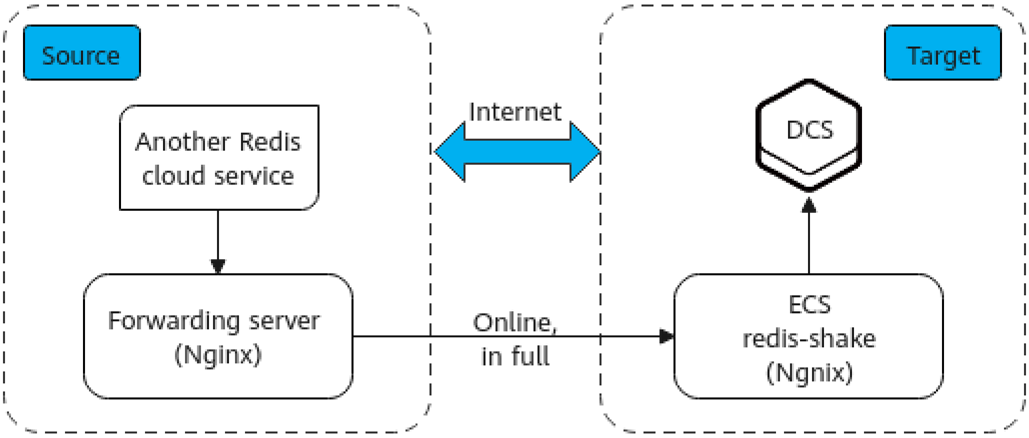
   If the data has not been fully imported, run the **flushall** or **flushdb** command to clear the cached data in the instance, and synchronize data again.

13.6.5 Online Migration from Another Cloud Using redis-shake

redis-shake is an open-source Redis migration tool. Its **rump** mode allows you to obtain the full data of a source Redis using the **SCAN** command and write the data to a target Redis. This migration solution does not involve the **SYNC** or **PSYNC** command and can be widely used for migration between self-built Redis and cloud Redis.

This section describes how to use the **rump** mode of redis-shake to migrate the full Redis data of another cloud service vendor at a time online to DCS.

Figure 13-12 Data flow in this solution



Prerequisites

- A **DCS Redis instance** has been created on the target cloud.
- An ECS has been created on the target cloud for running redis-shake.
- The ECS is in the same VPC as the DCS Redis instance and bound with an EIP.
- The **rump** mode does not support incremental data migration. To keep data consistency, stop writing data to the source Redis before migration.
- This solution applies only to same-database mapping and does not apply to inter-database mapping.
- If the source Redis has multiple databases (there are databases other than DB0), and your DCS instance is a cluster, this solution cannot be used. (Cluster DCS instances support only DB0.)

Procedure

Step 1 Install Nginx on the ECS and the source forwarding server. The following describes how to install Nginx on an ECS running CentOS 7.x. The commands vary depending on the OS.

1. Add Nginx to the Yum repository.

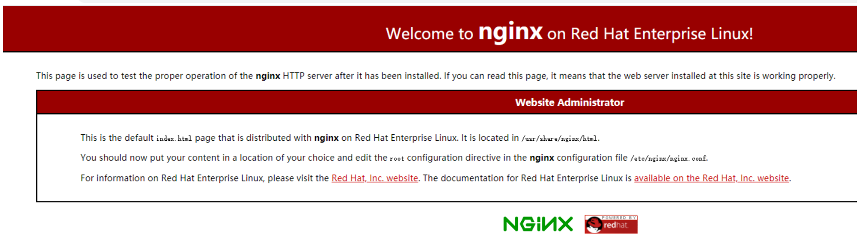
```
sudo rpm -Uvh http://nginx.org/packages/centos/7/noarch/RPMS/nginx-release-centos-7-0.el7.ngx.noarch.rpm
```

2. Check whether Nginx has been added successfully.

```
yum search nginx
```

3. Install Nginx.
   ```
   sudo yum install -y nginx
   ```
4. Install the stream module.
   ```
   yum install nginx-mod-stream --skip-broken
   ```
5. Start Nginx and set it to run automatically upon system startup.
   ```
   sudo systemctl start nginx.service
   sudo systemctl enable nginx.service
   ```
6. In the address box of a browser, enter the server address (the EIP of the ECS) to check whether Nginx is installed successfully.

   If the following page is displayed, Nginx has been installed successfully.



Welcome to **nginx** on Red Hat Enterprise Linux!

This page is used to test the proper operation of the **nginx** HTTP server after it has been installed. If you can read this page, it means that the web server installed at this site is working properly.

Website Administrator

This is the default index.html page that is distributed with **nginx** on Red Hat Enterprise Linux. It is located in /usr/share/nginx/html.

You should now put your content in a location of your choice and edit the root configuration directive in the **nginx** configuration file /etc/nginx/nginx.conf.

For information on Red Hat Enterprise Linux, please visit the Red Hat, Inc. website. The documentation for Red Hat Enterprise Linux is available on the Red Hat, Inc. website.

Step 2 Add the source forwarding server to the whitelist of the source Redis.

Step 3 Configure a security group for the source forwarding server.

1. Obtain the EIP of the ECS.
2. In the inbound rule of the security group of the source forwarding server, add the EIP of the ECS, and open the port that ECS's requests come through. The following takes port 6379 as an example.

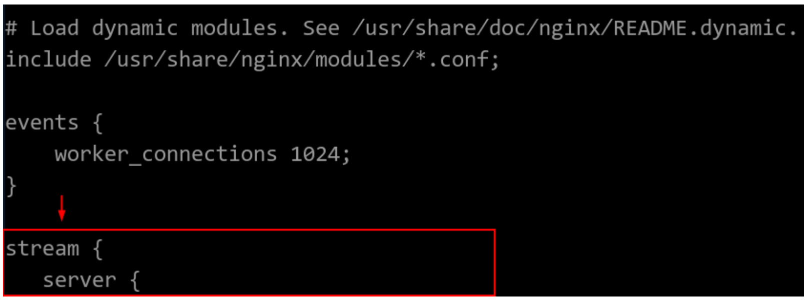
Step 4 Configure Nginx forwarding for the source forwarding server.

1. Log in to the Linux source forwarding server and run the following commands to open and modify the configuration file:
   ```
   cd /etc/nginx
   vi nginx.conf
   ```
2. Example forwarding configuration:
   ```
   stream {
       server {
           listen 6379;
           proxy_pass {source_instance_address}:{port};
       }
     }
   ```
   6379 is the listening port of the source forwarding server. *{source_instance_address}* and *{port}* are the connection address and port of the source Redis instance.

   This configuration allows you to access the source Redis through the local listening port 6379 of the source forwarding server.

   This configuration must be added exactly where it is shown in the following figure.

Figure 13-13 Configuration location



```
# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;

events {
    worker_connections 1024;
}

stream {
   server {
```

3. Restart Nginx.

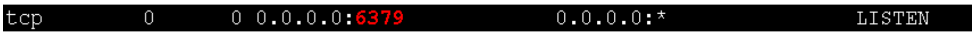
```
service nginx restart
```

4. Verify whether Nginx has been started.

```
netstat -an|grep 6379
```

If the port is being listened, Nginx has been started successfully.

Figure 13-14 Verification result



```
tcp        0      0 0.0.0.0:6379            0.0.0.0:*               LISTEN
```

Step 5 Configure Nginx forwarding for the ECS.

1. Log in to the Linux ECS and run the following commands to open and modify the configuration file:

   cd /etc/nginx

   vi nginx.conf

2. Configuration example:

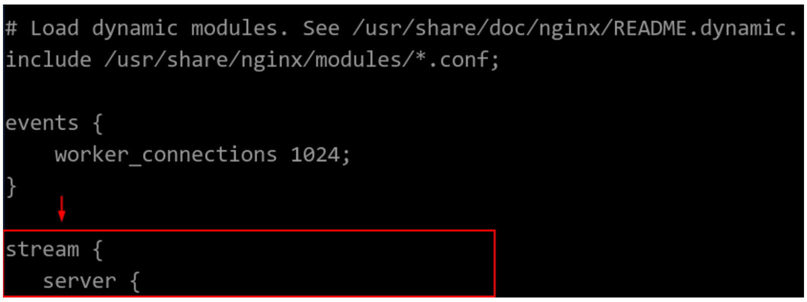
```
stream {
    server {
        listen 6666;
        proxy_pass {source_ecs_address}:6379;
    }
   }
```

6666 is ECS's listening port, *{source_ecs_address}* is the public IP address of the source forwarding server, and **6379** is the listening port of the source forwarding server Nginx.

This configuration allows you to access the source forwarding server through the local listening port 6666 of the ECS.

This configuration must be added exactly where it is shown in the following figure.

Figure 13-15 Configuration location



```
# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;

events {
    worker_connections 1024;
}

stream {
   server {
```

3. Restart Nginx.

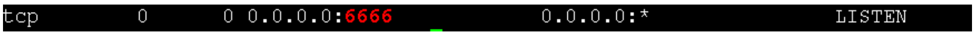
```
service nginx restart
```

4. Verify whether Nginx has been started.

```
netstat -an|grep 6666
```

If the port is being listened, Nginx has been started successfully.

Figure 13-16 Verification result



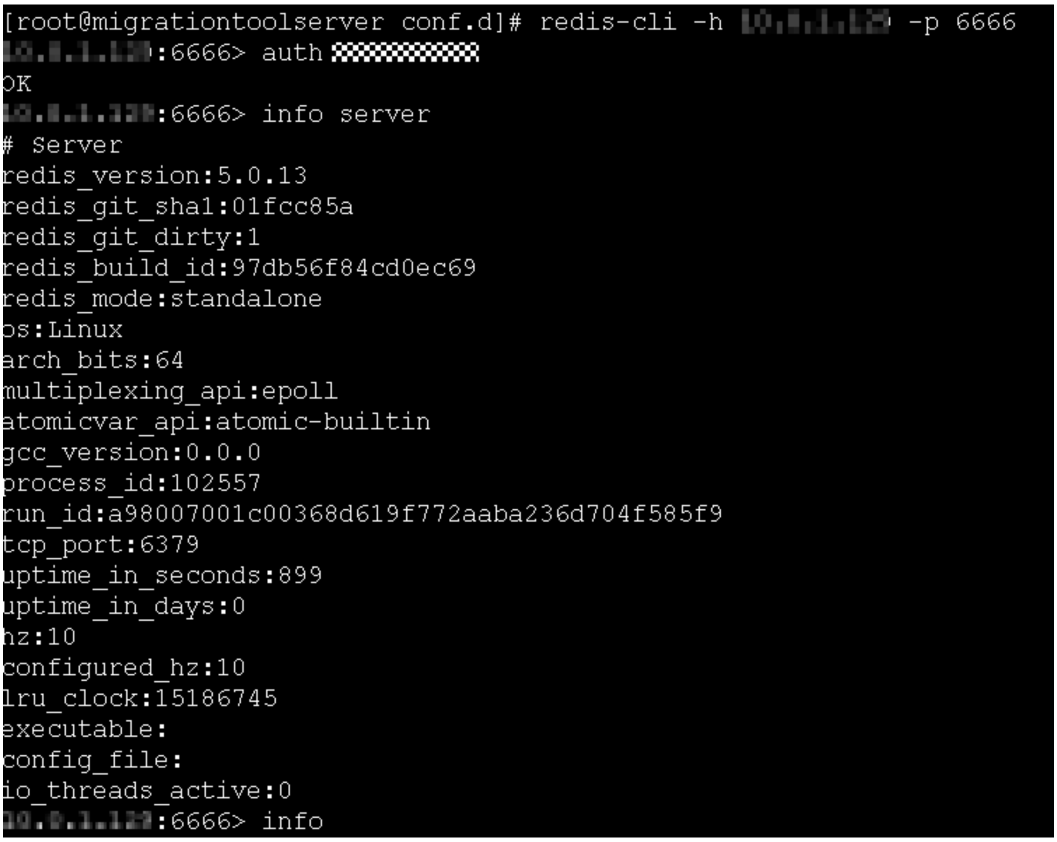
```
tcp        0      0 0.0.0.0:6666            0.0.0.0:*               LISTEN
```

Step 6 Run the following command on the ECS to test the network connection of port 6666:

```
redis-cli -h {target_ecs_address} -p 6666 -a {password}
```

{target_ecs_address} is the EIP of the ECS, **6666** is the listening port of the ECS, and *{password}* is the source Redis password. If there is no password, leave it blank.

Figure 13-17 Connection example



```
[root@migrationtoolserver conf.d]# redis-cli -h [redacted] -p 6666
[redacted]:6666> auth [redacted]
OK
[redacted]:6666> info server
# Server
redis_version:5.0.13
redis_git_sha1:01fcc85a
redis_git_dirty:1
redis_build_id:97db56f84cd0ec69
redis_mode:standalone
os:Linux
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:0.0.0
process_id:102557
run_id:a98007001c00368d619f772aaba236d704f585f9
tcp_port:6379
uptime_in_seconds:899
uptime_in_days:0
hz:10
configured_hz:10
lru_clock:15186745
executable:
config_file:
io_threads_active:0
[redacted]:6666> info
```

Step 7 Prepare the migration tool redis-shake.

1. Log in to the ECS.
2. Download redis-shake. Version 2.0.3 is used as an example. You can use **other redis-shake versions** as required.

```
wget https://github.com/tair-opensource/RedisShake/releases/download/release-v2.0.3-20200724/
redis-shake-v2.0.3.tar.gz
```

3. Decompress the redis-shake file.

```
tar -xvf redis-shake-v2.0.3.tar.gz
```

Step 8 Configure the redis-shake configuration file.

1. Go to the directory generated after the decompression.
   ```
   cd redis-shake-v2.0.3
   ```
2. Modify the **redis-shake.conf** configuration file.
   ```
   vim redis-shake.conf
   ```
   Modify the source Redis configuration.
   – source.type

     Type of the source Redis instance. Use **standalone** for single-node, master/standby, and Proxy Cluster, and **cluster** for cluster instances.
   – source.address

     EIP of the ECS and the mapped port of the source forwarding server (ECS's listening port 6666). Separate the EIP and port number with a colon (:).
   – source.password_raw

     Password of the source Redis instance. If no password is set, you do not need to set this parameter.

   Modify the target DCS configuration.
   – target.type

     Type of the DCS Redis instance. Use **standalone** for single-node, master/standby, and Proxy Cluster, and **cluster** for cluster instances.
   – target.address

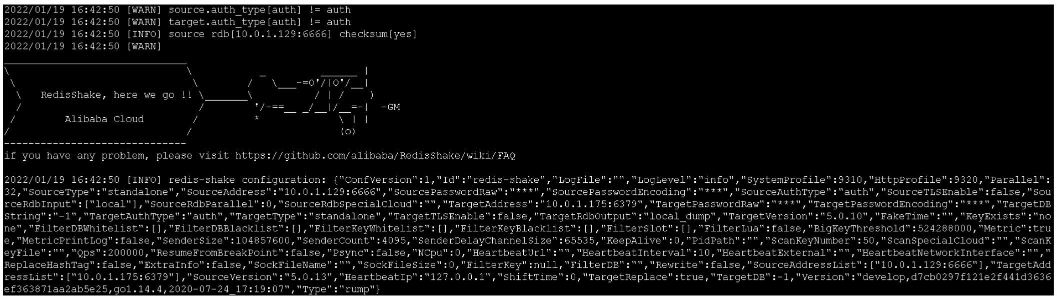
     Colon (:) separated connection address and port of the DCS Redis instance.
   – target.password_raw

     Password of the DCS Redis instance. If no password is set, you do not need to set this parameter.
3. Press **Esc** to exit the editing mode and enter **:wq!**. Press **Enter** to save the configuration and exit the editing interface.

Step 9 Run the following command to start redis-shake and migrate data in the **rump** (online in full) mode:

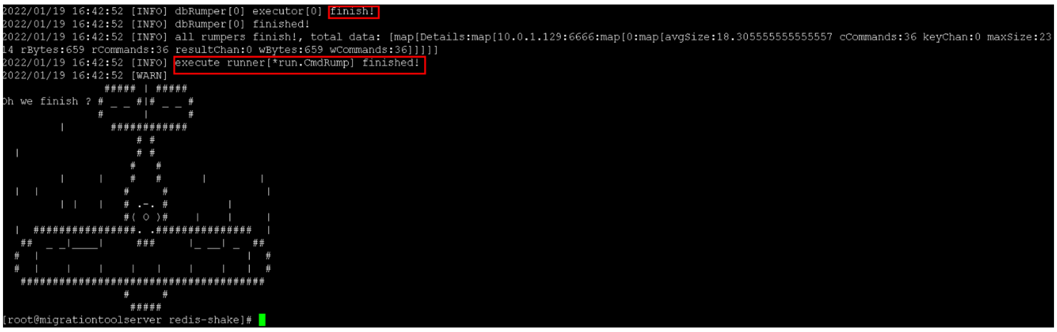
```
./redis-shake.linux -conf redis-shake.conf -type rump
```

Figure 13-18 Migration process



```
2022/01/19 16:42:50 [WARN] source.auth_type[auth] != auth
2022/01/19 16:42:50 [WARN] target.auth_type[auth] != auth
2022/01/19 16:42:50 [INFO] source rdb[10.0.1.129:6666] checksum[yes]
2022/01/19 16:42:50 [WARN]
______________________________
\                             \           _         ______ |
 \                             \        /   \___-=O'/|O'/__|
  \   RedisShake, here we go !! \_______\          / | /    )
  /                             /        '/-==__ _/__|/__=-|  -GM
 /        Alibaba Cloud        /         *             \ | |
/                             /                        (o)
------------------------------
if you have any problem, please visit https://github.com/alibaba/RedisShake/wiki/FAQ

2022/01/19 16:42:50 [INFO] redis-shake configuration: {"ConfVersion":1,"Id":"redis-shake","LogFile":"","LogLevel":"info","SystemProfile":9310,"HttpProfile":9320,"Parallel":
32,"SourceType":"standalone","SourceAddress":"10.0.1.129:6666","SourcePasswordRaw":"***","SourcePasswordEncoding":"***","SourceAuthType":"auth","SourceTLSEnable":false,"Sou
rceRdbInput":["local"],"SourceRdbParallel":0,"SourceRdbSpecialCloud":"","TargetAddress":"10.0.1.175:6379","TargetPasswordRaw":"***","TargetPasswordEncoding":"***","TargetDB
String":"-1","TargetAuthType":"auth","TargetType":"standalone","TargetTLSEnable":false,"TargetRdbOutput":"local_dump","TargetVersion":"5.0.10","FakeTime":"","KeyExists":"no
ne","FilterDBWhitelist":[],"FilterDBBlacklist":[],"FilterKeyWhitelist":[],"FilterKeyBlacklist":[],"FilterSlot":[],"FilterLua":false,"BigKeyThreshold":524288000,"Metric":tru
e,"MetricPrintLog":false,"SenderSize":104857600,"SenderCount":4095,"SenderDelayChannelSize":65535,"KeepAlive":0,"PidPath":"","ScanKeyNumber":50,"ScanSpecialCloud":"","ScanK
eyFile":"","Qps":200000,"ResumeFromBreakPoint":false,"Psync":false,"NCpu":0,"HeartbeatUrl":"","HeartbeatInterval":10,"HeartbeatExternal":"","HeartbeatNetworkInterface":"","
ReplaceHashTag":false,"ExtraInfo":false,"SockFileName":"","SockFileSize":0,"FilterKey":null,"FilterDB":"","Rewrite":false,"SourceAddressList":["10.0.1.129:6666"],"TargetAdd
ressList":["10.0.1.175:6379"],"SourceVersion":"5.0.13","HeartbeatIp":"127.0.0.1","ShiftTime":0,"TargetReplace":true,"TargetDB":-1,"Version":"develop,d7cb0297f121e2f441d3636
ef363871aa2ab5e25,go1.14.4,2020-07-24_17:19:07","Type":"rump"}
```

Figure 13-19 Migration result



```
2022/01/19 16:42:52 [INFO] dbRumper[0] executor[0] finish!
2022/01/19 16:42:52 [INFO] dbRumper[0] finished!
2022/01/19 16:42:52 [INFO] all rumpers finish!, total data: [map[Details:map[10.0.1.129:6666:map[0:map[avgSize:18.305555555555557 cCommands:36 keyChan:0 maxSize:23
14 rBytes:659 rCommands:36 resultChan:0 wBytes:659 wCommands:36]]]]]
2022/01/19 16:42:52 [INFO] execute runner[*run.CmdRump] finished!
2022/01/19 16:42:52 [WARN]
                ##### | #####
Oh we finish ? # _ _ #|# _ _ #
               #      |      #
         |       ############
                     # #
  |                  # #
                    #   #
         |     |    #   #       |        |
  |  |             #     #                |
         | |   |   # .-. #          |
                   #( O )#     |    |     |
  |  ################. .###############   |
   ##  _ _|____|     ###      |_ __| _  ##
  #  |                                 |  #
  #  |    |    |    |   |    |    |    |  #
   ######################################
                   #     #
                    #####
[root@migrationtoolserver redis-shake]#
```

Step 10 After the migration is complete, use redis-cli to connect to the source and target Redis instances to check whether the data is complete.

1. Connect to the source and target Redis instances, respectively.

   For details, see **Accessing a DCS Redis Instance Through redis-cli**.

2. Run the **info keyspace** command to check the values of **keys** and **expires**.
3. Calculate the differences between the values of **keys** and **expires** of the source Redis and the target Redis. If the differences are the same, the data is complete and the migration is successful.

Step 11 Delete the redis-shake configuration file.

----End

13.7 Migrating Data from DCS to Self-Hosted Redis

Scenario

You can use the online migration function of the DCS console to migrate DCS Redis instances to your self-hosted Redis. You can also export the DCS instance data to an RDB file and import it to local or self-hosted Redis.

Recommended Solutions

- Online migration on the DCS console

  For details, see **Online Migration of Self-Hosted Redis**. Select **Self-hosted Redis** and enter the target Redis address when configuring the target Redis.

- Use redis-cli or the DCS console to export the DCS instance data to an RDB file, and then use redis-shake to import the file to the target.

  For details about how to install and use redis-shake, see **Self-Hosted Redis Cluster Migration with redis-shake** and **redis-shake configuration instructions**.

- Rump

  This tool is recommended for online migration if possible. For details, see **Online Migration from Another Cloud Using Rump**.

14 FAQs

14.1 Instance Types/Versions

14.1.1 Comparing Versions

When creating a DCS Redis instance, you can select the cache engine version and the instance type.

- **Version**

  DCS supports Redis 3.0/4.0/5.0/6.0. **Table 14-1** describes the differences between these versions. For more information about Redis features, see **New Features of DCS for Redis 4.0** and **New Features of DCS for Redis 5.0**.

Table 14-1 Differences between Redis versions

Item	Redis 3.0	Redis 4.0 and Redis 5.0	Redis 6.0
Open-source compatibility	Redis 3.0.7	Redis 4.0.14 The latest DCS for Redis 5.0 is compatible with Redis 5.0.14. For existing users, see **How Do I View the Version of a DCS Redis Instance?**.	6.2.7
Instance deployment mode	Based on VMs	Containerized based on physical servers	Containerized based on physical servers

Item	Redis 3.0	Redis 4.0 and Redis 5.0	Redis 6.0
CPU architecture	x86 and Arm	x86 and Arm	x86
Time required for creating an instance	3–15 minutes, or 10–30 minutes for cluster instances.	8 seconds	8 seconds
QPS	100,000 QPS per node	100,000 QPS per node	150,000 QPS per node
Visualized data management	Not supported	Web CLI for connecting to Redis and managing data	Web CLI for connecting to Redis and managing data
Instance types	Single-node, master/standby, and Proxy Cluster	Single-node, master/standby, Redis Cluster	Single-node, master/standby
Instance total memory	Ranges from 2 GB to 1024 GB.	Regular specifications range from 2 GB to 1024 GB. Small specifications of 128 MB, 256 MB, 512 MB, and 1 GB are also available for single-node and master/standby instances.	4 GB, 8 GB, 16 GB, 32 GB, and 64 GB (128 MB, 256 MB, 512 MB, and 1 GB are additionally supported for single-node and master/standby instances)
Scale-up or scale-down	Online scale-up and scale-down	Online scale-up and scale-down	Online scale-up and scale-down

Item	Redis 3.0	Redis 4.0 and Redis 5.0	Redis 6.0
Backup and restoration	Supported for master/ standby and cluster instances	Supported for master/ standby and cluster instances	Master/Standby

NOTE

The underlying architectures vary by Redis version. Once a Redis version is chosen, it cannot be changed. For example, you cannot upgrade a DCS Redis 3.0 instance to Redis 4.0 or 5.0. If you require a higher Redis version, create a new instance that meets your requirements and then migrate data from the old instance to the new one.

- **Instance type**

  DCS provides single-node, master/standby, Proxy Cluster, and Redis Cluster instance types. For details about their architectures and application scenarios, see section "DCS Instance Types".

14.1.2 New Features of DCS for Redis 4.0

Compared with DCS for Redis 3.0, DCS for Redis 4.0 and later versions add support for the new features of open-source Redis and supports faster instance creation.

Instance deployment changed from the VM mode to physical server–based containerization mode. An instance can be created within 8 to 10 seconds.

Redis 4.0 provides the following new features:

1. New commands, such as **MEMORY** and **SWAPDB**
2. Lazyfree, delaying the deletion of large keys and reducing the impact of the deletion on system resources
3. Memory performance optimization, that is, active defragmentation

MEMORY Command

In Redis 3.0 and earlier versions, you can execute the **INFO MEMORY** command to learn only the limited memory statistics. Redis 4.0 introduces the **MEMORY** command to help you better understand Redis memory usage.

```
127.0.0.1:6379[8]> memory help
1) MEMORY <subcommand> arg arg ... arg. Subcommands are:
2) DOCTOR - Return memory problems reports.
3) MALLOC-STATS -- Return internal statistics report from the memory allocator.
4) PURGE -- Attempt to purge dirty pages for reclamation by the allocator.
5) STATS -- Return information about the memory usage of the server.
6) USAGE <key> [SAMPLES <count>] -- Return memory in bytes used by <key> and its value. Nested values
are sampled up to <count
> times (default: 5).
127.0.0.1:6379[8]>
```

usage

Enter **memory usage** ***[key]***. If the key exists, the estimated memory used by the value of the key is returned. If the key does not exist, **nil** is returned.

```
127.0.0.1:6379[8]> set dcs "DCS is an online, distributed, in-memory cache service compatible with Redis, and Memcached."
OK
127.0.0.1:6379[8]> memory usage dcs
(integer) 141
127.0.0.1:6379[8]>
```

NOTE

1. **usage** collects statistics on the memory usage of the value and the key, excluding the Expire memory usage of the key.

   ```
   // The following is verified based on Redis 5.0.2. Results may differ in other Redis versions.
   192.168.0.66:6379> set a "Hello, world!"
   OK
   192.168.0.66:6379> memory usage a
   (integer) 58
   192.168.0.66:6379> set abc "Hello, world!"
   OK
   192.168.0.66:6379> memory usage abc
   (integer) 60   //After the key name length changes, the memory usage also changes. This indicates that the usage statistics contain the usage of the key.
   192.168.0.66:6379> expire abc 1000000
   (integer) 1
   192.168.0.66:6379> memory usage abc
   (integer) 60   // After the expiration time is added, the memory usage remains unchanged. This indicates that the usage statistics do not contain the expire memory usage.
   192.168.0.66:6379>
   ```

2. For hashes, lists, sets, and sorted sets, the **MEMORY USAGE** command samples statistics and provides the estimated memory usage.

   Usage: **memory usage** *keyset* **samples** *1000*

   keyset indicates the key of a set, and *1000* indicates the number of samples.

stats

Returns the detailed memory usage of the current instance.

Usage: **memory stats**

```
127.0.0.1:6379[8]> memory stats
 1) "peak.allocated"
 2) (integer) 2412408
 3) "total.allocated"
 4) (integer) 2084720
 5) "startup.allocated"
 6) (integer) 824928
 7) "replication.backlog"
... ...
```

The following table describes the meanings of some return items.

Table 14-2 memory stats

Return Value	Description
peak.allocated	Peak memory allocated by the allocator during Redis instance running. It is the same as **used_memory_peak** of **info memory**.
total.allocated	The number of bytes allocated by the allocator. It is the same as **used_memory** of **info memory**

Return Value	Description
startup.allocated	Initial amount of memory consumed by Redis at startup in bytes.
replication.backlog	Size in bytes of the replication backlog. It is specified in the **repl-backlog-size** parameter. The default value is **1 MB**.
clients.slaves	The total size in bytes of all replicas overheads.
clients.normal	The total size in bytes of all clients overheads.
overhead.total	The sum of all overheads. **overhead.total** is the total memory **total.allocated** allocated by the allocator minus the actual memory used for storing data.
keys.count	The total number of keys stored across all databases in the server.
keys.bytes-per-key	Average number of bytes occupied by each key. Note that the overhead is also allocated to each key. Therefore, this value does not indicate the average key length.
dataset.bytes	Memory bytes occupied by Redis data, that is, **overhead.total** subtracted from **total.allocated**
dataset.percentage	The percentage of **dataset.bytes** out of the net memory usage.
peak.percentage	The percentage of peak.allocated out of **total.allocated**.
fragmentation	Memory fragmentation rate.

doctor

Usage: **memory doctor**

If the value of **used_memory** (**total.allocated**) is less than 5 MB, **MEMORY DOCTOR** considers that the memory usage is too small and does not perform further diagnosis. If any of the following conditions is met, Redis provides diagnosis results and suggestions:

1. The peak allocated memory is greater than 1.5 times of the current **total_allocated**, that is, **peak.allocated**/**total.allocated** > 1.5, indicating that the memory fragmentation rate is high, and that the RSS is much larger than **used_memory**.
2. The value of high fragmentation/fragmentation is greater than 1.4, indicating that the memory fragmentation rate is high.
3. The average memory usage of each normal client is greater than 200 KB, indicating that the pipeline may be improperly used or the Pub/Sub client does not process messages in time.
4. The average memory usage of each slave client is greater than 10 MB, indicating that the write traffic of the master is too high.

purge

Usage: **memory purge**

Executes the **jemalloc** internal command to release the memory. The released objects include the memory that is occupied but not used by Redis processes, that is, memory fragments.

NOTE

MEMORY PURGE applies only to the Redis instance that uses **jemalloc** as the allocator.

Lazyfree

Problem

Redis is single-thread. When a time-consuming request is executed, all requests are queued. Before the request is completed, Redis cannot respond to other requests. As a result, performance problems may occur. One of the time-consuming requests is deleting a large key.

Principle

The Lazyfree feature of Redis 4.0 avoids the blockage caused by deleting large keys, ensuring performance and availability.

When deleting a key, Redis asynchronously releases the memory occupied by the key. The key release operation is processed in the sub-thread of the background I/O (BIO).

Usage

1. Active deletion
   - **unlink**

     Similar to **DEL**, this command removes keys. If there are more than 64 elements to be deleted, the memory release operation is executed in an independent BIO thread. Therefore, the **UNLINK** command can delete a large key containing millions of elements in a short time.
   - **flushall/flushdb**

     An **ASYNC** option was added to **FLUSHALL** and **FLUSHDB** in order to let the entire dataset or a single database to be freed asynchronously.
2. Passive deletion: deletion of expired keys and eviction of large keys

   There are four scenarios for passive deletion and each scenario corresponds to a parameter. These parameters are disabled by default.

   ```
   lazyfree-lazy-eviction no // Whether to enable Lazyfree when the Redis memory usage reaches
   maxmemory and the eviction policy is set.
   lazyfree-lazy-expire no // Whether to enable Lazyfree when the key with TTL is going to expire.
   lazyfree-lazy-server-del no // An implicit DEL key is used when an existing key is processed.
   slave-lazy-flush no // Perform full data synchronization for the standby node. Before loading the RDB
   file of the master, the standby node executes the FLUSHALL command to clear its own data.
   ```

   NOTE

   To enable these configurations, contact technical support.

Other New Commands

1. **swapdb**

   Swaps two Redis databases.

   swapdb *dbindex1 dbindex2*

2. **zlexcount**

   Returns the number of elements in the sorted set.

   zlexcount *key min max*

Memory and Performance Optimization

1. Compared to before, the same amount of data can be stored with less memory.
2. Used memory can be defragmented and gradually evicted.

14.1.3 New Features of DCS for Redis 5.0

DCS for Redis 5.0 is compatible with the new features of the open-source Redis 5.0, in addition to all the improvements and new commands in Redis 4.0.

Stream Data Structure

Stream is a new data type introduced with Redis 5.0. It supports message persistence and multicast.

Figure 14-1 shows the structure of a Redis stream, which allows messages to be appended to the stream.

Streams have the following features:

1. A stream can have multiple consumer groups.
2. Each consumer group contains a **Last_delivered_id** that points to the last consumed item (message) in the consumer group.
3. Each consumer group contains multiple consumers. All consumers share the **last_delivered_id** of the consumer group. A message can be consumed by only one consumer.
4. **pending_ids** in the consumer can be used to record the IDs of items that have been sent to the client, but have not been acknowledged.
5. For detailed comparison between stream and other Redis data structures, see Table 14-3.

Figure 14-1 Stream data structure

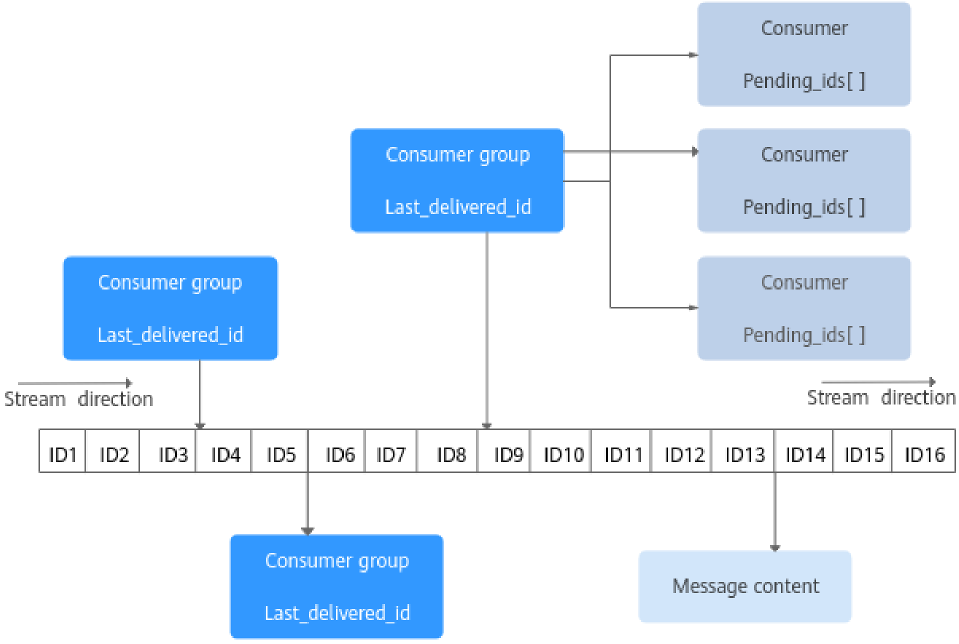


Table 14-3 Differences between streams and existing Redis data structures

Item	Stream	List, Pub/Sub, Zset
Complexity of seeking items	O(log(N))	List: O(N)
Offset	Supported. Each item has a unique ID. The ID is not changed as other items are added or evicted.	List: Not supported. If an item is evicted, the latest item cannot be located.
Persistence	Supported. Streams are persisted to AOF and RDB files.	Pub/Sub: Not supported.
Consumer group	Supported.	Pub/Sub: Not supported.
Acknowledgment	Supported.	Pub/Sub: Not supported.
Performance	Not related to the number of consumers.	Pub/Sub: Positively related to the number of clients.
Eviction	Streams are memory efficient by blocking to evict the data that is too old and using a radix tree and listpack.	Zset consumes more memory because it does not support inserting same items, blocking, or evicting data

Item	Stream	List, Pub/Sub, Zset
Randomly deleting items	Not supported.	Zset: Supported.

Stream commands

Stream commands are described below in the order they are used. For details, see **Table 14-4**.

1. Run the **XADD** command to add a stream item, that is, create a stream. The maximum number of messages that can be saved can be specified when adding the item.
2. Create a consumer group by running the **XGROUP** command.
3. A consumer uses the **XREADGROUP** command to consume messages.
4. After the consumption, the client runs the **XACK** command to confirm that the consumption is successful.

Figure 14-2 Stream commands

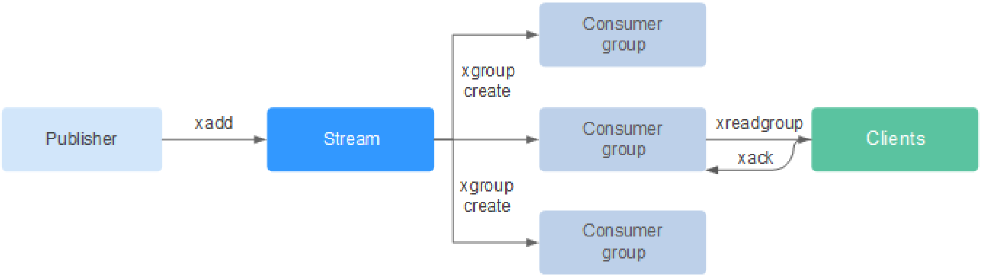


Table 14-4 Stream commands description

Command	Description	Syntax
XACK	Deletes one or multiple messages from the *pending entry list* (PEL) a consumer group of the stream.	XACK key group ID [ID ...]
XADD	Adds a specified entry to the stream at a specified key. If the key does not exist, running this command will result in a key to be automatically created based on the entry.	XADD key ID field string [field string ...]
XCLAIM	Changes the ownership of a pending message, so that the new owner is the consumer specified as the command argument.	XCLAIM key group consumer min-idle-time ID [ID ...] [IDLE ms] [TIME ms-unix-time] [RETRYCOUNT count] [FORCE] [JUSTID]

Command	Description	Syntax
XDEL	Removes the specified entries from a stream, and returns the number of entries deleted, that may be different from the number of IDs passed to the command in case certain IDs do not exist.	XDEL key ID [ID ...]
XGROUP	Manages the consumer groups associated with a stream You can use **XGROUP** to: ● Create a new consumer group associated with a stream. ● Destroy a consumer group. ● Remove a specified consumer from a consumer group. ● Set the consumer group *last delivery ID* to something else.	XGROUP [CREATE key groupname id-or-$] [SETID key id-or-$] [DESTROY key groupname] [DELCONSUMER key groupname consumername]
XINFO	Retrieves different information about the streams and associated consumer groups.	XINFO [CONSUMERS key groupname] key key [HELP]
XLEN	Returns the number of entries in a stream. If the specified key does not exist, **0** is returned, indicating an empty stream.	XLEN key
XPENDING	Obtains data from a stream through a consumer group. This command is the interface to inspect the list of pending messages in order to observe and understand what clients are active, what messages are pending to be consumed, or to see if there are idle messages.	XPENDING key group [start end count] [consumer]
XRANGE	Returns entries matching a given range of IDs.	XRANGE key start end [COUNT count]
XREAD	Reads data from one or multiple streams, only returning entries with an ID greater than the last received ID reported by the caller.	XREAD [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]
XREADGROUP	A special version of the **XREAD** command, which is used to specify a consumer group to read from.	XREADGROUP GROUP group consumer [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]

Command	Description	Syntax
XREVRANGE	This command is exactly like **XRANGE**, but with the notable difference of returning the entries in reverse order, and also taking the start-end range in reverse order.	XREVRANGE key end start [COUNT count]
XTRIM	Trims the stream to a specified number of items, if necessary, evicting old items (items with lower IDs).	XTRIM key MAXLEN [~] count

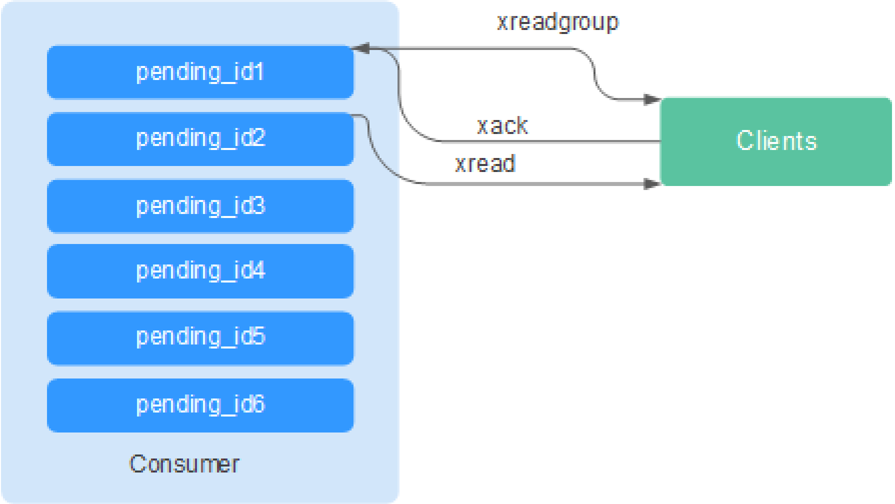
Message (stream item) acknowledgement

Compared with Pub/Sub, streams not only support consumer groups, but also message acknowledgement.

When a consumer invokes the **XREADGROUP** command to read or invokes the **XCLAIM** command to take over a message, the server does not know whether the message is processed at least once. Therefore, once having successfully processed a message, the consumer should invoke the **XACK** command to notify the stream so that the message will not be processed again. In addition, the message is removed from PEL and the memory will be released from the Redis server.

In some cases, such as network faults, the client does not invoke **XACK** after consumption. In such cases, the item ID is retained in PEL. After the client is reconnected, set the start message ID of **XREADGROUP** to 0-0, indicating that all PEL messages and messages after **last_id** are read. In addition, repeated message transmission must be supported when consumers consume messages.

Figure 14-3 Acknowledgment mechanism



Memory Usage Optimization

The memory usage of Redis 5.0 is optimized based on the previous version.

- Active defragmentation

  If a key is modified frequently and the value length changes constantly, Redis will allocate additional memory for the key. To achieve high performance, Redis uses the memory allocator to manage memory. Memory is not always freed up to the OS. As a result, memory fragments occur. If the fragmentation ratio (**used_memory_rss**/**used_memory**) is greater than 1.5, the memory usage is inefficient.

  To reduce memory fragments, properly plan and use cache data and standardize data writing.

  For Redis 3.0 and earlier versions, memory fragmentation problems are resolved by restarting the process regularly. It is recommended that the actual cache data does not exceed 50% of the available memory.

  For Redis 4.0, active defragmentation is supported, and memory is defragmented while online. In addition, Redis 4.0 supports manual memory defragmentation by running the **memory purge** command.

  For Redis 5.0, improved active defragmentation is supported with the updated Jemalloc, which is faster, more intelligent, and provides lower latency.

- HyperLogLog implementation improvements

  A HyperLogLog is a probabilistic data structure used to calculate the cardinality of a set while consuming little memory. Redis 5.0 improves HyperLogLog by further optimizing its memory usage.

  For example: the B-tree is efficient in counting, but consumes a lot of memory. By using HyperLogLog, a lot of memory can be saved. While the B-tree requires 1 MB memory for counting, HyperLogLog needs only 1 KB.

- Enhanced memory statistics

  The information returned by the **INFO** command is more detailed.

New and Better Commands

1. **Enhanced client management**

   - redis-cli supports cluster management.

     In Redis 4.0 and earlier versions, the **redis-trib** module needs to be installed to manage clusters.

     Redis 5.0 optimizes redis-cli, integrating all cluster management functions. You can run the **redis-cli --cluster help** command for more information.

   - The client performance is enhanced in frequent connection and disconnection scenarios.

     This optimization is valuable when your application needs to use short connections.

2. **Simpler use of sorted sets**

   ZPOPMIN and **ZPOPMAX** commands are added for sorted sets.

   - ZPOPMIN key [count]

     Removes and returns up to **count** members with the lowest scores in the sorted set stored at **key**. When returning multiple elements, the one with the lowest score will be the first, followed by the elements with higher scores.

– ZPOPMAX key [count]

  Removes and returns up to **count** members with the highest scores in the sorted set stored at **key**. When returning multiple elements, the one with the lowest score will be the first, followed by the elements with lower scores.

3. **More sub-commands added to the help command**

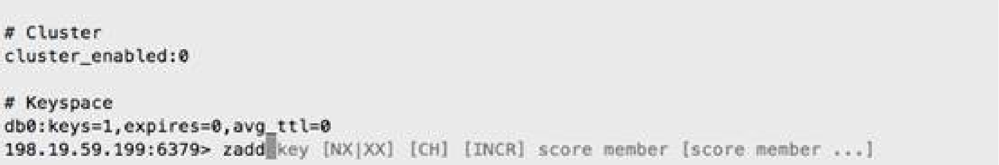
   The **help** command can be used to view help information, saving you the trouble of visiting **redis.io** every time. For example, run the following command to view the stream help information: **xinfo help**

```
127.0.0.1:6379> xinfo help
1) XINFO <subcommand> arg arg ... arg. Subcommands are:
2) CONSUMERS <key> <groupname>  -- Show consumer groups of group <groupname>.
3) GROUPS <key>                 -- Show the stream consumer groups.
4) STREAM <key>                 -- Show information about the stream.
5) HELP                         -- Print this help.
127.0.0.1:6379>
```

4. **redis-cli command input tips**

   After you enter a complete command, redis-cli displays a parameter tip to help you memorize the syntax format of the command.

   As shown in the following figure, run the **zadd** command, and redis-cli displays **zadd** syntax in light color.



```
# Cluster
cluster_enabled:0

# Keyspace
db0:keys=1,expires=0,avg_ttl=0
198.19.59.199:6379> zadd key [NX|XX] [CH] [INCR] score member [score member ...]
```

RDB Storing LFU and LRU Information

In Redis 5.0, storage key eviction policies **LRU** and **LFU** were added to the RDB snapshot file.

- FIFO: First in, first out. The earliest stored data is evicted first.
- LRU: Least recently used. Data that is not used for a long time is evicted first.
- LFU: Least frequently used. Data that is least frequently used is evicted first.

NOTE

The RDB file format of Redis 5.0 is modified and is backward compatible. Therefore, if a snapshot is used for migration, data can be migrated from the earlier Redis versions to Redis 5.0, but cannot be migrated from the Redis 5.0 to the earlier versions.

14.1.4 What Are the CPU Specifications of DCS Instances?

When using DCS, you only need to pay attention to critical indicators such as QPS, bandwidth, and memory. You do not need to be concerned about CPU specifications.

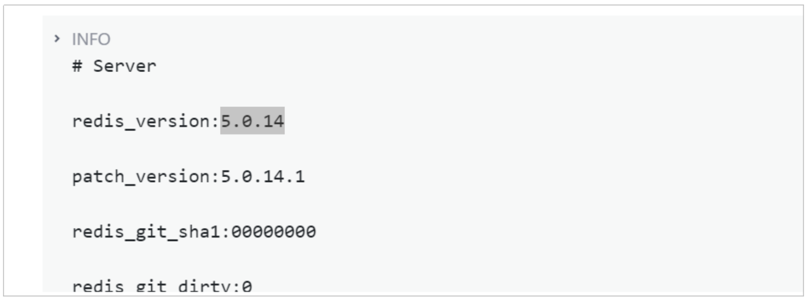
DCS for Redis is based on open-source Redis. Open-source Redis uses a single main thread to process commands, so only one CPU core is used on each Redis node. Increasing the memory of DCS Redis instances does not change CPU specifications.

Due to this restriction, **you can use a cluster instance and add shards to achieve higher CPU performance**. Each node in a cluster uses one vCPU by default.

14.1.5 How Do I View the Version of a DCS Redis Instance?

Connect to the instance and run the **INFO** command.

Figure 14-4 Querying instance information



```
› INFO
  # Server

  redis_version:5.0.14

  patch_version:5.0.14.1

  redis_git_sha1:00000000

  redis_git_dirty:0
```

14.2 Client and Network Connection

14.2.1 Security Group Configurations

DCS helps you control access to your DCS instances in the following ways, depending on the deployment mode:

- To control access to DCS Redis 3.0 and Memcached instances, you can use security groups. Whitelists are not supported. Security group operations are described in this section.
- To control access to DCS Redis 4.0/5.0/6.0 instances, you can use whitelists. Security groups are not supported. Whitelist operations are described in **Managing IP Address Whitelist**.

The following describes how to configure security groups for **intra-VPC access** to DCS Redis 3.0 and Memcached instances.

Intra-VPC Access to DCS Redis 3.0 and Memcached Instances

An ECS can communicate with a DCS instance if they belong to the same VPC and security group rules are configured correctly.

In addition, you must configure correct rules for the security groups of both the ECS and DCS instance so that you can access the instance through your client.

- If the ECS and DCS instance are configured with the same security group, network access in the group is not restricted by default.
- If the ECS and DCS instance are configured with different security groups, add security group rules to ensure that the ECS and DCS instance can access each other.

NOTE

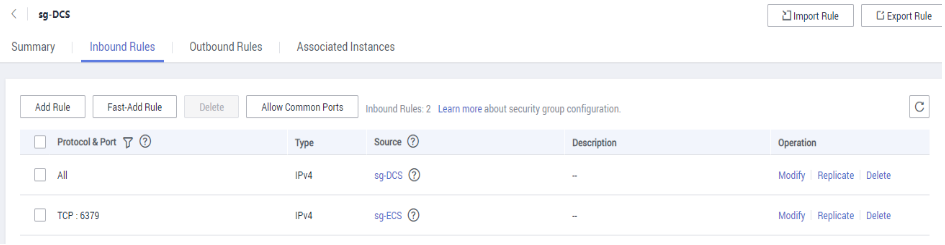
- Suppose that the ECS on which the client runs belongs to security group **sg-ECS**, and the DCS instance that the client will access belongs to security group **sg-DCS**.
- Suppose that the port number of the DCS service is 6379.
- The remote end is a security group or an IP address.

a. Configuring security group for the ECS.

Add the following outbound rule to allow the ECS to access the DCS instance. Skip this rule if there are no restrictions on the outbound traffic.

b. Configuring security group for the DCS instance.

To ensure that your client can access the DCS instance, add the following inbound rule to the security group configured for the DCS instance:



Inbound Rules: 2 Learn more about security group configuration.

□ Protocol & Port	Type	Source	Description	Operation
□ All	IPv4	sg-DCS	--	Modify | Replicate | Delete
□ TCP : 6379	IPv4	sg-ECS	--	Modify | Replicate | Delete

NOTICE

In the inbound rule of the security group configured for the DCS instance, set the remote end to an IP address in the same CIDR block as the subnet.

To prevent ECSs bound with same security group as the DCS instance from being attacked by Redis vulnerabilities, exercise caution when using **0.0.0.0/0**.

14.2.2 Does DCS Support Access over Public Networks?

No. DCS instances cannot be access at their EIPs over public networks. To ensure security, the ECS that serves as a client and the DCS instance that the client will access must belong to the same VPC.

14.2.3 Does DCS Support Cross-VPC Access?

Cross-VPC means the client and the instance are not in the same VPC.

Generally, VPCs are isolated from each other and ECSs cannot access DCS instances that belong to a different VPC from these ECSs.

However, by establishing VPC peering connections between VPCs, ECSs can access single-node and master/standby DCS instances across VPCs.

When using VPC peering connections to access DCS instances across VPCs, adhere to the following rules:

- If network segments 172.16.0.0/12 to 172.16.0.0/24 are used during DCS instance creation, the client cannot be in any of the following network segments: 192.168.1.0/24, 192.168.2.0/24, and 192.168.3.0/24.
- If network segments 192.168.0.0/16 to 192.168.0.0/24 are used during DCS instance creation, the client cannot be in any of the following network segments: 172.31.1.0/24, 172.31.2.0/24, and 172.31.3.0/24.
- If network segments 10.0.0.0/8 to 10.0.0.0/24 are used during DCS instance creation, the client cannot be in any of the following network segments: 172.31.1.0/24, 172.31.2.0/24, and 172.31.3.0/24.

For more information about VPC peering connection, see "VPC Peering Connection" in the *Virtual Private Cloud User Guide*.

NOTICE

Cluster DCS Redis instances do not support cross-VPC access. ECSs in a VPC cannot access cluster DCS instances in another VPC by using VPC peering connections.

14.2.4 Why Is "(error) NOAUTH Authentication required" Displayed When I Access a DCS Redis Instance?

This is because you have enabled password-free access for the instance. To prevent the error message from appearing, do not enter any password.

14.2.5 What Should I Do If Access to DCS Fails After Server Disconnects?

Analysis: If persistent connections ("pconnect" in Redis terminology) or connection pooling is used and connections are closed after being used for connecting to DCS instances, errors will be returned at attempts to reuse the connections.

Solution: When using pconnect or connection pooling, do not close the connection after the end of a request. If the connection is dropped, re-establish it.

14.2.6 Why Do Requests Sometimes Time Out in Clients?

Occasional timeout errors are normal because of network connectivity and client timeout configurations.

You are advised to include reconnection operations into your service code to avoid service failure if a single request fails.

If a connection request times out, check if AOF persistence has been enabled. To avoid blocking, ensure that AOF has been enabled.

If timeout errors occur frequently, contact O&M personnel.

14.2.7 What Should I Do If an Error Is Returned When I Use the Jedis Connection Pool?

The error message that will possibly be displayed when you use the Jedis connection pool is as follows:

```
redis.clients.jedis.exceptions.JedisConnectionException: Could not get a resource from the pool
```

If this error message is displayed, check whether your instance is running properly. If it is running properly, perform the following checks:

Step 1 Network

1. Check the IP address configurations.

   Check whether the IP address configured on the Jedis client is the same as the subnet address configured for your DCS instance.

2. Test the network.

   Use the ping command and telnet on the client to test the network.

   – If the network cannot be pinged:

     For intra-VPC access to a DCS Redis 3.0 or Memcached instance, ensure that the client and your DCS instance belong to the same VPC and security group, or the security group of your DCS instance allows access through port 6379. For details, see **Security Group Configurations**.

   – If the IP address can be pinged but telnet failed, restart your instance. If the problem persists after the restart, contact technical support.

Step 2 Check the number of connections.

Check whether the number of established network connections exceeds the upper limit configured for the Jedis connection pool. If the number of established connections approaches the configured upper limit, restart the DCS service and check whether the problem persists. If the number of established connections is far below the upper limit, continue with the following checks.

In Unix or Linux, run the following command to query the number of established network connections:

netstat -an | grep 6379 | grep ESTABLISHED | wc -l

In Windows, run the following command to query the number of established network connections:

netstat -an | find "6379" | find "ESTABLISHED" /C

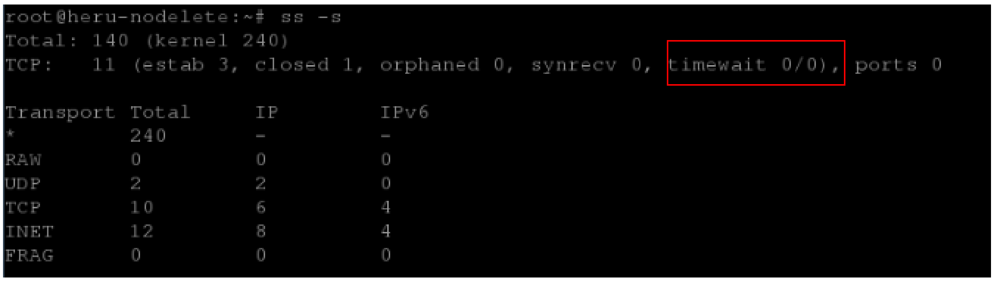
Step 3 Check the JedisPool code.

If the number of established connections approaches the upper limit, determine whether the problem is caused by service concurrency or incorrect usage of JedisPool.

When using JedisPool, you must call **jedisPool.returnResource()** or **jedis.close()** (recommended) to release the resources after you call **jedisPool.getResource()**.

Step 4 Check the number of TIME_WAIT connections.

Run the **ss -s** command to check whether there are too many **TIME_WAIT** connections on the client.



```
root@heru-nodelete:~# ss -s
Total: 140 (kernel 240)
TCP:   11 (estab 3, closed 1, orphaned 0, synrecv 0, timewait 0/0), ports 0

Transport Total     IP        IPv6
*         240       -         -
RAW       0         0         0
UDP       2         2         0
TCP       10        6         4
INET      12        8         4
FRAG      0         0         0
```

If there are too many **TIME_WAIT** connections, modify the kernel parameters by running the **/etc/sysctl.conf** command as follows:

```
##Uses cookies to prevent some SYN flood attacks when the SYN waiting queue overflows.
net.ipv4.tcp_syncookies = 1
##Reuses TIME_WAIT sockets for new TCP connections.
net.ipv4.tcp_tw_reuse = 1
##Enables quick reclamation of TIME_WAIT sockets in TCP connections.
net.ipv4.tcp_tw_recycle = 1
##Modifies the default timeout time of the system.
net.ipv4.tcp_fin_timeout = 30
```

After the modification, run the **/sbin/sysctl -p** command for the modification to take effect.

Step 5 If the problem persists after you perform the preceding checks, perform the following steps.

Capture packets and send packet files along with the time and description of the exception to technical support for analysis.

Run the following command to capture packets:

tcpdump -i eth0 tcp and port 6379 -n -nn -s 74 -w dump.pcap

In Windows, you can also install the Wireshark tool to capture packets.

NOTE

Replace the NIC name to the actual one.

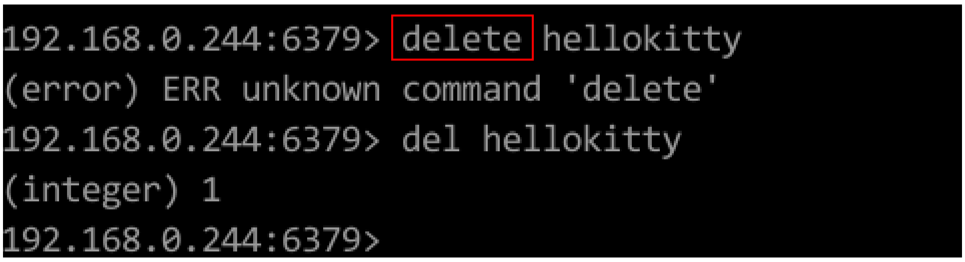
----End

14.2.8 Why Is "ERR unknown command" Displayed When I Access a DCS Redis Instance Through a Redis Client?

The possible causes are as follows:

1. The command is spelled incorrectly.

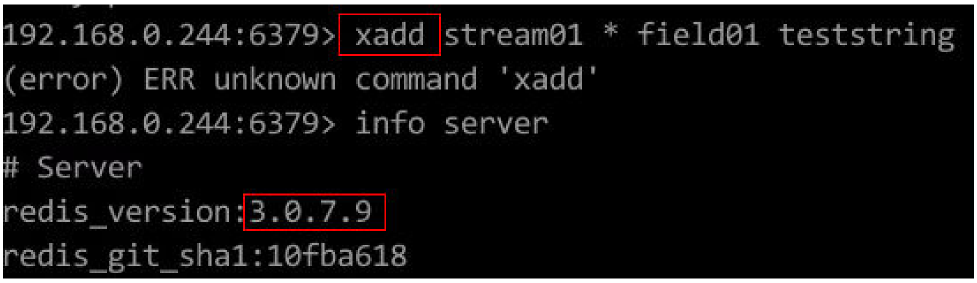
   As shown in the following figure, the error message is returned because the correct command for deleting a string should be **del**.



```
192.168.0.244:6379> delete hellokitty
(error) ERR unknown command 'delete'
192.168.0.244:6379> del hellokitty
(integer) 1
192.168.0.244:6379>
```

2. A command available in a higher Redis version is run in a lower Redis version.

   As shown in the following figure, the error message is returned because a stream command (available in Redis 5.0) is run in Redis 3.0.



```
192.168.0.244:6379> xadd stream01 * field01 teststring
(error) ERR unknown command 'xadd'
192.168.0.244:6379> info server
# Server
redis_version:3.0.7.9
redis_git_sha1:10fba618
```

3. Some commands are disabled.

   DCS Redis instance interfaces are fully compatible with the open-source Redis in terms of data access. However, for ease of use and security purposes, some operations cannot be initiated through Redis clients. For details about disabled commands, see **Command Compatibility**.

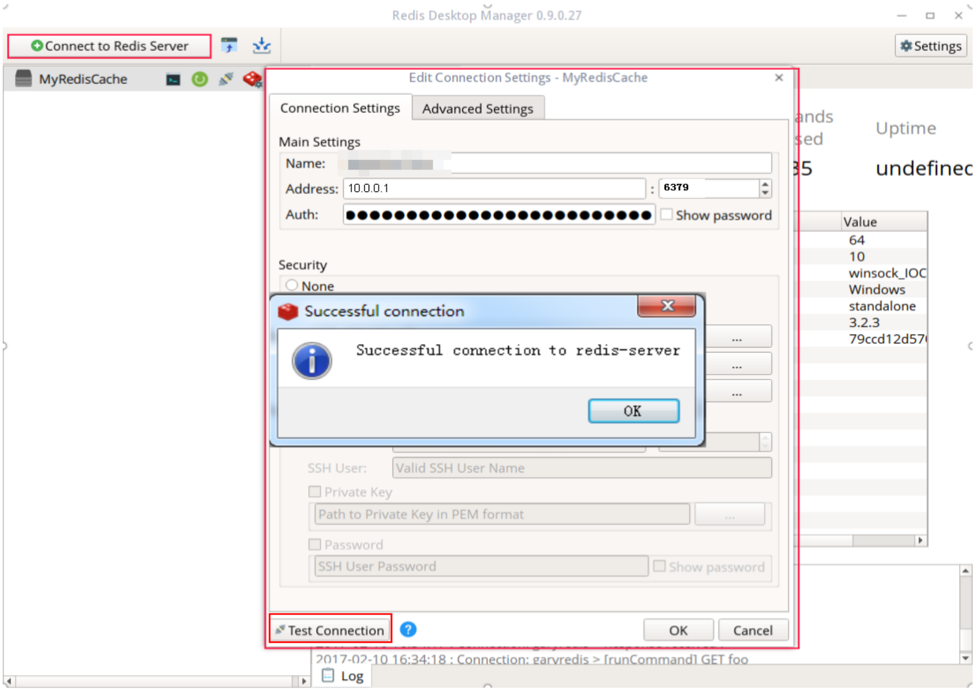
14.2.9 How Do I Access a DCS Redis Instance Through Redis Desktop Manager?

You can access a DCS Redis instance through the Redis Desktop Manager within a VPC.

1. Enter the address, port number (6379), and authentication password of the DCS instance you want to access.
2. Click **Test Connection**.

   The system displays a success message if the connection is successful.

Figure 14-5 Accessing a DCS Redis instance through Redis Desktop Manager over the intranet



Edit Connection Settings - MyRedisCache

Main Settings

Name: [redacted]

Address: 10.0.0.1 : 6379

Auth: ●●●●●●●●●●●●●●●●●●●●●●●●●● □ Show password

Security

○ None

Successful connection

Successful connection to redis-server

SSH User: Valid SSH User Name

□ Private Key

Path to Private Key in PEM format

□ Password

SSH User Password □ Show password

2017-02-10 16:34:18 : Connection: garyredis > [runCommand] GET foo

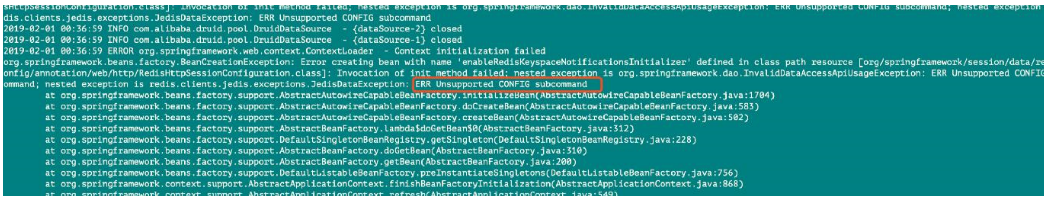
NOTE

When accessing a cluster DCS instance, the Redis command is run properly, but an error message may display on the left because DCS clusters are based on Codis, which differs from the native Redis in terms of the **INFO** command output.

14.2.10 What If "ERR Unsupported CONFIG subcommand" is Displayed in SpringCloud?

By using DCS Redis instances, Spring Session can implement session sharing. When interconnecting with Spring Cloud, the following error information is displayed:

Figure 14-6 Spring Cloud error information



```
sHttpSessionConfiguration.class]: Invocation of init method failed; nested exception is org.springframework.dao.InvalidDataAccessApiUsageException: ERR Unsupported CONFIG subcommand; nested exception
dis.clients.jedis.exceptions.JedisDataException: ERR Unsupported CONFIG subcommand
2019-02-01 00:36:59 INFO com.alibaba.druid.pool.DruidDataSource  - {dataSource-2} closed
2019-02-01 00:36:59 INFO com.alibaba.druid.pool.DruidDataSource  - {dataSource-1} closed
2019-02-01 00:36:59 ERROR org.springframework.web.context.ContextLoader  - Context initialization failed
org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'enableRedisKeyspaceNotificationsInitializer' defined in class path resource [org/springframework/session/data/re
onfig/annotation/web/http/RedisHttpSessionConfiguration.class]: Invocation of init method failed; nested exception is org.springframework.dao.InvalidDataAccessApiUsageException: ERR Unsupported CONFIG
ommand; nested exception is redis.clients.jedis.exceptions.JedisDataException: ERR Unsupported CONFIG subcommand
        at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:1704)
        at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:583)
        at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:502)
        at org.springframework.beans.factory.support.AbstractBeanFactory.lambda$doGetBean$0(AbstractBeanFactory.java:312)
        at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:228)
        at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:310)
        at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:200)
        at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:756)
        at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:868)
        at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:549)
```

For security purposes, DCS does not support the **CONFIG** command initiated by a client. You need to perform the following steps:

1. On the DCS console, set the value of the **notify-keyspace-event** parameter to **Egx** for a DCS Redis instance.
2. Add the following content to the XML configuration file of the Spring framework:

   <util:constant

   static-field="org.springframework.session.data.redis.config.ConfigureRedisAction.NO_OP"/>
3. Modify the related Spring code. Enable the **ConfigureRedisAction.NO_OP** bean component to forbid a client to invoke the **CONFIG** command.
   ```
   @Bean
   public static ConfigureRedisAction configureRedisAction() {
       return ConfigureRedisAction.NO_OP;
   }
   ```

For more information, see the **Spring Session Documentation**.

NOTICE

Session sharing is supported by single-node and master/standby DCS Redis instances, but not by cluster DCS Redis instances.

14.2.11 Is a Password Required for Accessing an Instance? How Do I Set a Password?

- A DCS Redis instance can be access with or without a password. You can directly access a DCS Redis instance through a Redis client without setting a password. However, for security purposes, you are advised to set a password for authentication and verification whenever possible. The password must be set when you create the instance.

- A DCS Memcached instance can be access with or without a password. You can select any Memcached client that supports the Memcached text protocol and binary protocol based on specific application features. The password must be set when you create the instance.
- To change the Redis instance access mode, or change or reset a password, see **Managing Passwords**.

14.2.12 What Should Be Noted When Using Redis for Pub/ Sub?

Pay attention to the following issues when using Redis for pub/sub:

- Your client must process messages in a timely manner.

  Your client subscribes to a channel. If it does not receive messages in a timely manner, DCS instance messages may be overstocked. If the size of accumulated messages reaches the threshold (32 MB by default) or remains at a certain level (8 MB by default) for a certain period of time (1 minute by default), your client will be automatically disconnected to prevent server memory exhaustion.
- Your client must support connection re-establishment in case of disconnection.

  In the event of a disconnection, you need to run the **subscribe** or **psubscribe** command on your client to subscribe to a channel again. Otherwise, your client cannot receive messages.
- Do not use pub/sub in scenarios with high message reliability requirements.

  The Redis pub/sub is not a reliable messaging system. Messages that are not retrieved will be discarded when your client is disconnected or a master/ standby switchover occurs.

14.2.13 How Do I Troubleshoot Redis Connection Failures?

Preliminary checks:

- Check the connection address.

  Obtain the connection address from the instance basic information page on the DCS console.
- Check the instance password.

  If the instance password is incorrect, the port can still be accessed but the authentication will fail.
- Check the port.

  Port 6379 is the default port used in intra-VPC access to a DCS Redis instance.
- Check if the maximum bandwidth has been reached.

  If the bandwidth reaches the maximum bandwidth for the corresponding instance specifications, Redis connections may time out.
- For a DCS Redis 3.0 instance, check the inbound access rules of the security group.

Intra-VPC access: If the Redis client and the Redis instance are bound with different security groups, allow inbound access over port 6379 for the security group of the instance.

For details, see **Security Group Configurations**.

- For a DCS Redis 4.0/5.0/6.0 instance, check the whitelist configuration.

  If the instance has a whitelist, ensure that the client IP address is included in the whitelist. Otherwise, the connection will fail.

  For details, see **Managing IP Address Whitelist**.

  If the client IP address has changed, add the new IP address to the whitelist.

- Check the configuration parameter **notify-keyspace-events**.

  Set **notify-keyspace-events** to **Egx**.

Further checks:

- Jedis connection pool error
- Error "Read timed out" or "Could not get a resource from the pool"

  Check if the **KEYS** command has been used. This command consumes a lot of resources and can easily block Redis. Instead, use the **SCAN** command and avoid executing the command frequently.

14.2.14 What Can I Do If Error "Cannot assign requested address" Is Returned When I Access Redis Using connect?

Symptom

Error message "Cannot assign requested address" is returned when you access Redis using **connect**.

Analysis

Applications that encounter this error typically use php-fpm and phpredis. In high-concurrency scenarios, a large number of TCP connections are in the TIME-WAIT state. As a result, the client cannot allocate new ports and the error message will be returned.

Solutions

- Solution 1: Use **pconnect** instead of **connect**.

  Using **pconnect** reduces the number of TCP connections and prevents connections from being re-established for each request, and therefore reduces latency.

  When using **connect**, the code for connecting to Redis is as follows:

  ```
  $redis->connect('${Hostname}',${Port});
  $redis->auth('${Inst_Password}');
  ```

  Replace **connect** with **pconnect**, and the code becomes:

  ```
  $redis->pconnect('${Hostname}', ${Port}, 0, NULL, 0, 0, ['auth' => ['${Inst_Password}']]);
  ```

NOTE

- Replace the connection parameters in the example with actual values. *${Hostname}*, *${Port}*, and *${Inst_Password}* are the connection address, port number, and password of the Redis instance, respectively.
- phpredis must be v5.3.0 or later. You are advised to use this **pconnect** initialization mode to avoid NOAUTH errors during disconnection.

- Solution 2: Modify the **tcp_max_tw_buckets** parameter of the ECS where the client is located.

  In this solution, the ports used by TIME-WAIT connections are reused. However, if retransmission occurs between the ECS and the backend service, the connection may fail. Therefore, the **pconnect** solution is recommended.

  a. Connect to the ECS where the client is located

  b. Run the following command to check the **ip_local_port_range** and **tcp_max_tw_buckets** parameters:

     sysctl net.ipv4.tcp_max_tw_buckets net.ipv4.ip_local_port_range

     Information similar to the following is displayed:

     ```
     net.ipv4.tcp_max_tw_buckets = 262144
     net.ipv4.ip_local_port_range = 32768  61000
     ```

  c. Run the following command to set the **tcp_max_tw_buckets** parameter to a value smaller than the value of **ip_local_port_range**:

     sysctl -w net.ipv4.tcp_max_tw_buckets=10000

Generally, solution 1 is recommended. In special scenarios (for example, the service code involves too many components and is difficult to change), solution 2 can be used to meet high concurrency requirements.

14.2.15 Connection Pool Selection and Recommended Jedis Parameter Settings

Advantages of the Jedis Connection Pool

The comparison between Lettuce and Jedis is as follows:

- Lettuce
  - Lettuce does not perform connection keepalive detection. If an abnormal connection exists in the connection pool, an error is reported when requests time out.
  - Lettuce does not implement connection pool validation such as **testOnBorrow**. As a result, connections cannot be validated before being used.
- Jedis
  - Jedis implements connection pool validation using **testOnBorrow**, **testWhileIdle**, and **testOnReturn**.

    If **testOnBorrow** is enabled, connection validation is performed when connections are being borrowed, which has the highest reliability but affects the performance (detection is performed before each Redis request).
  - **testWhileIdle** can be used to detect idle connections. If the threshold is set properly, abnormal connections in the connection pool can be

removed in time to prevent service errors caused by abnormal connections.

- If a connection becomes abnormal before the idle connection check, the service that uses the connection may report an error. You can specify the **timeBetweenEvictionRunsMillis** parameter to control the check interval.

Therefore, Jedis has better exception handling and detection capabilities and is more reliable than Lettuce in scenarios where there are connection exceptions and network jitters.

Recommended Jedis Connection Pool Parameter Settings

Table 14-5 Recommended Jedis connection pool parameter settings

Parameter	Description	Recommended Setting
maxTotal	Maximum number of connections	Set this parameter based on the number of HTTP threads of the web container and reserved connections. Assume that the **maxConnections** parameter of the Tomcat Connector is set to **150** and each HTTP request may concurrently send two requests to Redis, you are advised to set this parameter to at least 400 (150 x 2 + 100). **Limit**: The value of **maxTotal** multiplied by the number of client nodes (CCE containers or service VMs) must be less than the maximum number of connections allowed for a single DCS Redis instance. For example, if **maxClients** of a master/standby DCS Redis instance is 10,000 and **maxTotal** of a single client is 500, the maximum number of clients is 20.
maxIdle	Maximum number of idle connections	Set this parameter to the value of **maxTotal**.

Parameter	Description	Recommended Setting
minIdle	Minimum number of idle connections	Generally, you are advised to set this parameter to 1/X of **maxTotal**. For example, the recommended value is **100**. In performance-sensitive scenarios, you can set this parameter to the value of **maxIdle** to prevent the impact caused by frequent connection quantity changes. For example, set this parameter to **400**.
maxWaitMillis	Maximum waiting time for obtaining a connection, in milliseconds	The recommended maximum waiting time for obtaining a connection from the connection pool is the maximum tolerable timeout of a single service minus the timeout for command execution. For example, if the maximum tolerable HTTP timeout is 15s and the timeout of Redis requests is 10s, set this parameter to 5s.
timeout	Command execution timeout, in milliseconds	This parameter indicates the maximum timeout for running a Redis command. Set this parameter based on the service logic. Generally, you are advised to set this timeout to longer than 210 ms to ensure network fault tolerance. For special detection logic or environment exception detection, you can adjust this timeout to seconds.

Parameter	Description	Recommended Setting
minEvictableIdleTimeMillis	Idle connection eviction time, in milliseconds. If a connection is not used for a period longer than this, it will be released.	If you do not want the system to frequently re-establish disconnected connections, set this parameter to a large value (xx minutes) or set this parameter to **–1** and check idle connections periodically.
timeBetweenEvictionRunsMillis	Interval for detecting idle connections, in milliseconds	The value is estimated based on the number of idle connections in the system. For example, if this interval is set to 30s, the system detects connections every 30s. If an abnormal connection is detected within 30s, it will be removed. Set this parameter based on the number of connections. If the number of connections is too large and this interval is too short, request resources will be wasted. If there are hundreds of connections, you are advised to set this parameter to 30s. The value can be dynamically adjusted based on system requirements.
testOnBorrow	Indicates whether to check the connection validity using the **ping** command when borrowing connections from the resource pool. Invalid connections will be removed.	If your service is extremely sensitive to connections and the performance is acceptable, you can set this parameter to **True**. Generally, you are advised to set this parameter to **False** to enable idle connection detection.

Parameter	Description	Recommended Setting
testWhileIdle	Indicates whether to use the **ping** command to monitor the connection validity during idle resource monitoring. Invalid connections will be destroyed.	True
testOnReturn	Indicates whether to check the connection validity using the **ping** command when returning connections to the resource pool. Invalid connections will be removed.	False
maxAttempts	Number of connection retries when JedisCluster is used	Recommended value: 3–5. Default value: **5**. Set this parameter based on the maximum timeout intervals of service APIs and a single request. The maximum value is **10**. If the value exceeds **10**, the processing time of a single request is too long, blocking other requests.

14.3 Redis Usage

14.3.1 What Are Shard and Replica Quantities?

Shard

A **shard** is a management unit in Redis clusters. Each shard corresponds to a redis-server process. A cluster consists of multiple shards. Each shard has multiple slots. Data is distributedly stored in the slots. Shards increase cache capacity and concurrent connections.

Each cluster instance consists of multiple shards. By default, each shard is a master/standby instance with two replicas. The number of shards is equal to the number of master nodes in a cluster instance.

Replica

A replica refers to a **node** of a DCS instance. It can be a master node or a standby node. A single-replica instance has no standby node. A two-replica instance has one master node and one standby node. For example, if the number of replicas is set to three for a master/standby instance, the instance has one master node and two standby nodes.

Number of Replicas and Shards of Different Instance Types

- **Single-node**: Each instance has only one node (one Redis process). If the Redis process is faulty, DCS starts a new Redis process for the instance.
- **Master/Standby**:Each instance has one shard. Each shard has one master node, and one or more standby nodes. If a master node is faulty, a master/standby switchover will be performed to restore the service.
- **Cluster**: Each instance has multiple shards. By default, each shard is a master/standby instance with two replicas. For example, if a cluster instance has three shards and two replicas, each shard has two nodes (one master node and one standby node).

Instance Type	Shards	Replicas	Load Balancing	IP Addresses
Single-node	1	1 (only)	-	1
Master/Standby	1	2 (default) DCS Redis 4.0/5.0 Master/Standby instances support 2 to 10 replicas. Master/Standby instances of other versions support only 2 replicas.	Not supported	Same as the number of replicas
Proxy Cluster	Multiple	2 (not customizable)	Supported	1

Instance Type	Shards	Replicas	Load Balancing	IP Addresses
Redis Cluster	Multiple	2 (default) DCS Redis 4.0/5.0 Redis Cluster instances support 1 to 5 replicas. Redis Cluster 6.0 instances support 1 to 2 replicas.	Not supported	Number of replicas x Number of shards

14.3.2 Why Is CPU Usage of a DCS Redis Instance 100%?

- Possible cause 1:

  The service QPS is so high that the CPU usage spikes to 100%.
- Possible cause 2:

  You have run commands that consume a lot of resources, such as **KEYS**. This will make CPU usage spike and can easily trigger a master/standby switchover.
- Possible cause 3:

  Redis rewrite was triggered, increasing CPU usage.

For details, see **Troubleshooting High CPU Usage of a DCS Redis Instance**.

14.3.3 Can I Change the VPC and Subnet for a DCS Redis Instance?

No. Once an instance is created, its VPC and subnet cannot be changed. If you want to use a different set of VPC and subnet, create a same instance and specify a desired set of VPC and subnet. After the new instance is created, you can migrate data from the old instance to the new instance by following the **data migration instructions**.

14.3.4 Why Aren't Security Groups Configured for DCS Redis 4.0/5.0/6.0 Instances?

Currently, DCS Redis 4.0/5.0/6.0 instances use VPC endpoints and do not support security groups. You can configure whitelists instead. For details, see **Managing IP Address Whitelist**.

To allow access only from specific IP addresses to a DCS Redis instance, add the IP addresses to the instance whitelist.

If no whitelists are added to the instance whitelist or the whitelist function is disabled, all IP addresses that can communicate with the VPC can access the instance.

14.3.5 Do DCS Redis Instances Limit the Size of a Key or Value?

- The maximum allowed size of a key is 512 MB.

  To reduce memory usage and facilitate key query, ensure that each key does not exceed 1 KB.
- The maximum allowed size of a string is 512 MB.
- The maximum allowed size of a Set, List, or Hash is 512 MB.

  In essence, a Set is a collection of Strings; a List is a list of Strings; a Hash contains mappings between string fields and string values.

Prevent the client from constantly writing large values in Redis. Otherwise, network transmission efficiency will be lowered and the Redis server would take a longer time to process commands, resulting in higher latency.

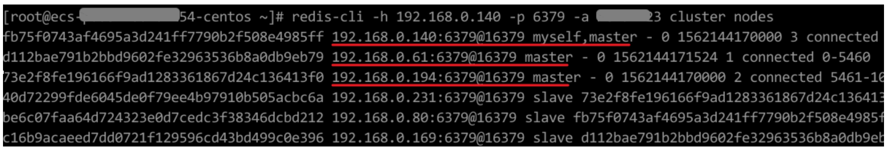
14.3.6 Can I Obtain the Addresses of the Nodes in a Cluster DCS Redis Instance?

Cluster DCS Redis 3.0 instances (Proxy Cluster type) are used in the same way that you use single-node or master/standby instances. You do not need to know the backend node addresses.

For a cluster DCS Redis 4.0 or later instance (Redis Cluster type), run the **CLUSTER NODES** command to obtain node addresses:

redis-cli -h *{redis_address}* -p *{redis_port}* -a *{redis_password}* cluster nodes

In the output similar to the following, obtain the IP addresses and port numbers of all the master nodes.



```
[root@ecs-[redacted]54-centos ~]# redis-cli -h 192.168.0.140 -p 6379 -a [redacted]3 cluster nodes
fb75f0743af4695a3d241ff7790b2f508e4985ff 192.168.0.140:6379@16379 myself,master - 0 1562144170000 3 connected
d112bae791b2bbd9602fe32963536b8a0db9eb79 192.168.0.61:6379@16379 master - 0 1562144171524 1 connected 0-5460
73e2f8fe196166f9ad1283361867d24c136413f0 192.168.0.194:6379@16379 master - 0 1562144170000 2 connected 5461-10
40d72299fde6045de0f79ee4b97910b505acbc6a 192.168.0.231:6379@16379 slave 73e2f8fe196166f9ad1283361867d24c136413
be6c07faa64d724323e0d7cedc3f38346dcbd212 192.168.0.80:6379@16379 slave fb75f0743af4695a3d241ff7790b2f508e4985f
c16b9acaeed7dd0721f129596cd43bd499c0e396 192.168.0.169:6379@16379 slave d112bae791b2bbd9602fe32963536b8a0db9eb
```

14.3.7 Why Is Available Memory Smaller Than Instance Cache Size?

DCS Redis 3.0 and Memcached instances are deployed on VMs and some memory is reserved for system overheads. This problem will not occur on other instance versions.

14.3.8 Does DCS for Redis Support Read/Write Splitting?How Do I Configure Read/Write Splitting for a Redis Cluster Instance?

Configuration

- For a **Redis Cluster instance**, you can query all master and replica nodes by running the **CLUSTER NODES** command. The client will connect to replicas and configure read-only access on them.

Run the following command to query cluster nodes:

```
redis-cli -h {redis_address} -p {redis_port} -a {redis_password} cluster nodes
```

Read-only configuration on replicas is achieved through the **READONLY** command.

14.3.9 Does DCS for Redis Support Multiple Databases?

Both single-node and master/standby DCS Redis instances support multiple databases. By default, single-node and master/standby DCS instances can read and write data in 256 databases (databases numbering 0–255). The default database is DB0.

Redis Cluster DCS instances do not support multi-DB. There is only one database (DB0).

14.3.10 Does DCS for Redis Support Redis Clusters?

Yes. DCS for Redis 4.0 and later support Redis Clusters. DCS for Redis 3.0 supports Proxy Clusters.

14.3.11 What Is Sentinel?

Overview

High availability in Redis is implemented through Sentinel. Sentinel helps you defend against certain types of faults without manual intervention, and complete tasks such as monitoring, notification, and client configuration. For details, see the **Redis official website**.

Principles

Redis Sentinel is a distributed system where multiple Sentinel processes work together. It has the following advantages:

1. Fault detection is performed only when multiple Sentinels agree that a master node is unavailable, which reduces the possibility of false positives.
2. Even if some Sentinel processes are faulty, the Sentinel system can still work properly to prevent faults.

On a higher level, there is a larger distributed system consisting of Sentinels, Redis master and replica nodes, and clients connected to Sentinels and Redis.

Functions

- Monitoring: Sentinel continuously checks whether the master and replica nodes are working properly.
- Notification: If a node is faulty, Sentinel can notify the system administrator or other computer programs by calling an API.
- Automatic failover: If the master node is abnormal, Sentinel starts a failover to promote a replica to master. Other replicas replicate data from the new master node. Applications that use the Redis instance will be notified that they should connect to the new address.

- Client configuration: Sentinel serves as the authoritative source for client service discovery. Clients connect to Sentinel and requests the address of the master Redis node that is responsible for specific services. If a failover occurs, Sentinels delivers the new address.

14.3.12 Does DCS for Redis Support Sentinel?

Cluster instances and master/standby DCS Redis 4.0 and later instances support Sentinels. Sentinels monitor the running status of both the master and standby nodes of a master/standby instance and each shard of a cluster instance. If the master node becomes faulty, a failover will be performed.

DCS for Redis 3.0 does not support Redis Sentinel. Instead, it uses keepalive to monitor master and replica nodes and to manage failovers.

14.3.13 What Is the Default Data Eviction Policy?

Data is evicted from cache based on a user-defined space limit in order to make space for new data. For details, see the **Redis official website**. You can **view or change the eviction policy** by configuring an instance parameter on the DCS console.

Eviction Policies Supported by DCS Redis Instances

When **maxmemory** is reached, you can select one of the following eight eviction policies:

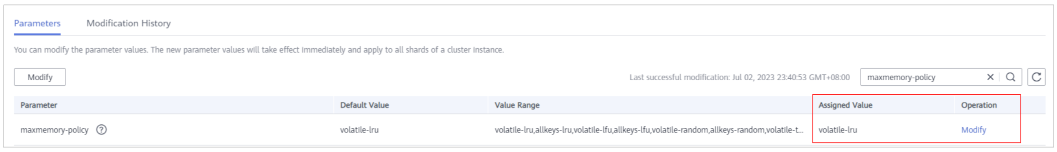
- **noeviction**: When the memory limit is reached, DCS instances return errors to clients and no longer process write requests and other requests that could result in more memory to be used. However, **DEL** and a few more exception requests can continue to be processed.
- **allkeys-lru**: DCS instances try to evict the least recently used keys first, in order to make space for new data.
- **volatile-lru**: DCS instances try to evict the least recently used keys with an expire set first, in order to make space for new data.
- **allkeys-random**: DCS instances recycle random keys so that new data can be stored.
- **volatile-random**: DCS instances evict random keys with an expire set, in order to make space for new data.
- **volatile-ttl**: DCS instances evict keys with an expire set, and try to evict keys with a shorter time to live (TTL) first, in order to make space for new data.
- **allkeys-lfu**: DCS instances evict the least frequently used keys from all keys.
- **volatile-lfu**: DCS instances evict the least frequently used keys with an **expire** field from all keys.

NOTE

If no key can be recycled, **volatile-lru**, **volatile-random**, and **volatile-ttl** are the same as **noeviction**. For details, see the description of **noeviction**.

Viewing or Changing Eviction Policies

You can view or change the eviction policy with the **maxmemory-policy** parameter.



You can modify the parameter values. The new parameter values will take effect immediately and apply to all shards of a cluster instance.

Last successful modification: Jul 02, 2023 23:40:53 GMT+08:00

Parameter	Default Value	Value Range	Assigned Value	Operation
maxmemory-policy	volatile-lru	volatile-lru,allkeys-lru,volatile-lfu,allkeys-lfu,volatile-random,allkeys-random,volatile-t...	volatile-lru	Modify

14.3.14 What Should I Do If an Error Occurs in Redis Exporter?

Start the Redis exporter using the CLI. Based on the output, check for errors and troubleshoot accordingly.

```
[root@ecs-swk /]./redis_exporter -redis.addr 192.168.0.23:6379
INFO[0000] Redis Metrics Exporter V0.15.0   build date:2018-01-19-04:08:01 sha1:
a0d9ec4704b4d35cd08544d395038f417716a03a
  Go:go1.9.2
INFO[0000] Providing metrics at :9121/metrics
INFO[0000] Connecting to redis hosts: []string{192.168.0.23:6379}
INFO[0000] Using alias:[]string{""}
```

14.3.15 How Can I Secure My DCS Redis Instances?

Redis is one of the most powerful and widely used open-source cache technologies. However, the open-source Redis does not have robust security features of its own. It is vulnerable to malicious Internet attacks, possibly causing data breaches.

To secure your DCS Redis instances, consider taking the following advice:

- Network connection configurations
  a. Encrypt sensitive data.

     Sensitive data must be encrypted before being stored.
  b. Configure access rules for your security groups.

     Security groups and VPCs are designed for securing network access. Allow access over as few ports as possible to avoid risks.
  c. Configure ECS firewalls.

     Configure firewall filtering rules for the ECS where your client runs.
  d. Set the instance password.
  e. Configure a whitelist.
- redis-cli usage
  a. Hide the password.

     Problem: If the **-a <password>** option is used, the password may show up when the **ps** command is run.

     Solution: Modify the Redis source code. Hide the password immediately after starting redis-cli by calling the main function.
  b. Disable sudo in running scripts.

     Problem: Parameters for starting redis-cli contain sensitive patterns related to the password, which may show up when the **ps** command is run and may be logged.

Solution: Access the instance by calling APIs (or through redis-py in Python). Do not allow switching to the **dbuser** user using sudo in redis-cli.

14.3.16 Why Is Redisson Distributed Lock Not Supported by DCS Proxy Cluster Redis 3.0 Instances?

Redisson implements lock acquisition and unlocking in the following process:

1. Redisson lock acquisition and unlocking are implemented by running Lua scripts.
2. During lock acquisition, the **EXISTS**, **HSET**, **PEXPIRE**, **HEXISTS**, **HINCRBY**, **PEXPIRE**, and **PTTL** commands must be executed in the Lua script.
3. During unlocking, the **EXISTS**, **PUBLISH**, **HEXISTS**, **PEXPIRE**, and **DEL** commands must be executed in the Lua script.

In a proxy-based cluster, the proxy processes **PUBLISH** and **SUBSCRIBE** commands and forwards requests to the Redis server. The **PUBLISH** command cannot be executed in the Lua script.

As a result, Proxy Cluster DCS Redis 3.0 instances do not support Redisson distributed locks. To use Redisson, resort to Redis 4.0 or 5.0 instead.

14.3.17 Can I Customize or Change the Port for Accessing a DCS Instance?

You cannot customize or change the port for accessing a DCS Redis 3.0 or Memcached instance. You can customize (during instance creation) but cannot change the port for accessing a DCS Redis 4.0, 5.0, or 6.0 instance.

- Redis 3.0

  Intra-VPC access: port 6379.
- Memcached

  Use port 11211 for intra-VPC access. Public access is not supported.
- Redis 4.0 and later

  You can specify a port (ranging from 1 to 65535) or use the default port (6379) for accessing an instance. If no port is specified, the default port will be used.

If the instance and the client use different security groups, you must configure access rules for the security groups, allowing access through the specified port. For details, see **Security Group Configurations**.

14.3.18 Can I Modify the Connection Addresses for Accessing a DCS Instance?

After a DCS instance is created, its intra-VPC connection addresses cannot be modified.

To use a different IP address, you must create a new instance and manually specify an IP address. After the instance is created, migrate the data from the old instance to the new instance.

For details about accessing DCS instances through clients, see **Accessing a DCS Redis Instance Through redis-cli**.

14.3.19 Why Do I Fail to Delete an Instance?

Possible causes and solutions:

- The instance is not in the **Running** state.

  Only instances in the **Running** state can be deleted.
- The instance fails to be created.

  Check whether the instance fails to be created by clicking the icon or number after **Instance Creation Failures** above the instance list. If yes, delete it.

14.3.20 Does DCS Support Cross-AZ Deployment?

Master/Standby and cluster DCS Redis instances and DCS Memcached instances can be deployed across availability zones (AZs).

- If instance nodes in an AZ are faulty, nodes in other AZs will not be affected. The standby node automatically becomes the master node to continue to operate, ensuring disaster recovery (DR).
- Cross-AZ deployment does not compromise the speed of data synchronization between the master and standby nodes.

14.3.21 Why Does It Take a Long Time to Start a Cluster DCS Instance?

Possible cause: When a cluster instance is started, status and data are synchronized between the nodes of the instance. If a large amount of data is continuously written into the instance before the synchronization is complete, the synchronization will be prolonged and the instance remains in the **Starting** state. After the synchronization is complete, the instance enters the **Running** state.

Solution: Start writing data to an instance only after the instance has been started.

14.3.22 Does DCS for Redis Provide Backend Management Software?

No. If you wish to query Redis configurations and usage information, use redis-cli. If you wish to monitor DCS Redis instance metrics, go to the Cloud Eye console. For details on how to configure and view the metrics, see **Monitoring**.

14.3.23 Can I Recover Data from Deleted DCS Instances?

If a DCS instance is automatically deleted or manually deleted through the Redis client, its data cannot be retrieved. If you have backed up the instance, you can restore its data from the backup. However, the restoration will overwrite the data written in during the period from the backup and the restoration.

You can restore backup data to a master/standby cluster, or instance through **Backups & Restorations** on the DCS console. For details, see **Restoring a DCS Instance**.

If a DCS instance is deleted, the instance data and its backup will also be deleted. Before deleting an instance, you can download the backup files of the instance for permanent local storage and can also migrate them to a new instance if you need to restore the data. For details about how to download the backup data, see **Downloading a Backup File**

14.3.24 Does DCS for Redis Support SSL Encrypted Transmission?

By default, SSL is disabled for DCS Redis 6.0 instances. To enable it, see **Transmitting DCS Redis Data with Encryption Using SSL**.

DCS Redis for 3.0/4.0/5.0 only support plaintext transmission. They do not support SSL encrypted transmission.

14.3.25 Why Is Available Memory of Unused DCS Instances Less Than Total Memory and Why Is Memory Usage of Unused DCS Instances Greater Than Zero?

For DCS Redis 3.0 instances and Memcached instances, the available memory is less than the total memory because some memory is reserved for system overhead and data persistence (supported by master/standby instances). DCS instances use a certain amount of memory for Redis-server buffers and internal data structures. This is why memory usage of unused DCS instances is greater than zero. This problem will not occur on other instance versions.

14.3.26 How Do I Check Redis Memory Usage?

Currently, DCS for Redis provides the following memory-related metrics:

Table 14-6 DCS Redis instance metrics

Metric ID	Metric Name	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	Monitored object: DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric Name	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
used_memory	Used Memory	Number of bytes used by the Redis server Unit: KB, MB, or byte (configurable on the console)	≥ 0	Monitored object: DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_dataset	Used Memory Dataset	Dataset memory that the Redis server has used Unit: KB, MB, or byte (configurable on the console)	≥ 0	Monitored object: DCS Redis instance Only Redis 4.0 and later Dimension: dcs_instance_id	1 minute
used_memory_dataset_perc	Used Memory Dataset Ratio	Percentage of data memory that Redis has used to the total used memory Unit: %	0–100%	Monitored object: DCS Redis instance Only Redis 4.0 and later Dimension: dcs_instance_id	1 minute
used_memory_rss	Used Memory RSS	Resident set size (RSS) memory that the Redis server has used, which is the memory that actually resides in the memory, including all stack and heap memory but not swapped-out memory Unit: KB, MB, or byte (configurable on the console)	≥ 0	Monitored object: DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric Name	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
memory_frag_ratio	Memory Fragmentation Ratio	Current memory fragmentation, which is the ratio between **used_memory_rss**/ **used_memory**.	≥ 0	Monitored object: DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_peak	Used Memory Peak	Peak memory consumed by Redis since the Redis server last started Unit: KB, MB, or byte (configurable on the console)	≥ 0	Monitored object: DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_lua	Used Memory Lua	Number of bytes used by the Lua engine Unit: KB, MB, or byte (configurable on the console)	≥ 0	Monitored object: DCS Redis instance Dimension: dcs_instance_id	1 minute

14.3.27 Why Is the Capacity or Performance of a Shard of a Redis Cluster Instance Overloaded When That of the Instance Is Still Below the Bottleneck?

Redis Cluster uses a special data sharding method. **Every key is part of a hash slot, which is held by a node in the cluster.** To compute what is the hash slot of a given key:

1. Take the CRC16 of the key modulo 16384.
2. Based on the mapping between hash slots and shards, connections are redirected to the right node for data read and write operations.

Therefore, keys are not evenly distributed to each shard of an instance. If a shard contains a big key or a hot key, the capacity or performance of the shard will be overloaded, but the load on other shards is still low. As a result, the capacity or performance bottleneck of the entire instance is not reached.

14.3.28 Does DCS Support External Extensions, Plug-ins, or Modules?

No. DCS for Redis does not support external extensions, plug-ins, or modules. There is no plan for supporting modules.

14.3.29 Why Is "Error in execution" Returned When I Access Redis?

Symptom: "Error in execution; nested exception is io.lettuce.core.RedisCommandExecutionException: OOM command not allowed when used memory > 'maxmemory'" is returned during a Redis connection.

Analysis: An out-of-memory (OOM) error indicates that the maximum memory is exceeded. In the error information, the "maxmemory" parameter indicates the maximum memory configured on the Redis server.

If the memory usage of the Redis instance is less than 100%, the memory of the node where data is written may have reached the maximum limit. Connect to each node in the cluster by running **redis-cli -h <redis_ip> -p 6379 -a <redis_password> -c --bigkeys**. When connecting to a replica node, run the **READONLY** command before running the **bigkeys** command.

14.3.30 Why Does a Key Disappear in Redis?

Normally, Redis keys do not disappear. If a key is missing, it may have expired, been evicted, or been deleted.

Perform the following checks one by one:

1. Check whether the key has expired.
2. View the monitoring information and check whether eviction was triggered.
3. Run the **INFO** command on the server side to check whether the key has been deleted.

14.3.31 Why Does an OOM Error Occur During a Redis Connection?

Symptom

"Error in execution; nested exception is io.lettuce.core.RedisCommandExecutionException: OOM command not allowed when used memory > 'maxmemory'" is returned during a Redis connection.

Fault Locating

An out-of-memory (OOM) error indicates that the maximum memory is exceeded. In the error information, the **maxmemory** parameter indicates the maximum memory configured on the Redis server.

If the memory usage of the Redis instance is less than 100%, the memory of the node where data is written may have reached the maximum limit. Connect to each node in the cluster by running **redis-cli -h <redis_ip> -p 6379 -a <redis_password> -c --bigkeys**. When connecting to a replica node, run the **READONLY** command before running the **bigkeys** command.

14.3.32 What Clients Can I Use for Redis Cluster in Different Programming Languages?

The following table compares Redis Cluster and Proxy Cluster in DCS.

Table 14-7 Comparing Redis Cluster and Proxy Cluster

Item	Redis Cluster	Proxy Cluster
Redis compatibility	High	Medium
Client compatibility	Medium (The cluster mode must be enabled on the client.)	High
Costs	High	Medium
Latency	Low	Medium
Read/write splitting	Native support (client SDK configuration)	Implemented by using proxies
Performance	High	Medium

Redis Cluster does not use proxies, and therefore delivers lower latency and higher performance. However, Redis Cluster instances are based on the open-source Redis Cluster protocol, so their client compatibility is poorer than that of Proxy Cluster instances.

The following table lists clients that can be used for Redis Cluster.

Table 14-8 Clients that can be used for Redis Cluster

Language	Client	Reference Document
Java	Jedis	**https://github.com/xetorthio/jedis#jedis-cluster**
Java	Lettuce	**https://github.com/lettuce-io/lettuce-core/wiki/Redis-Cluster**
PHP	php redis	**https://github.com/phpredis/phpredis#readme**

Language	Client	Reference Document
Go	Go Redis	Redis Cluster: **https://pkg.go.dev/github.com/go-redis/redis/v8#NewClusterClient** Proxy Cluster, single-node, or master/standby: **https://pkg.go.dev/github.com/go-redis/redis/v8#NewClient**
Python	redis-py-cluster	**https://github.com/Grokzen/redis-py-cluster#usage-example**
C	hiredis-vip	**https://github.com/vipshop/hiredis-vip?_ga=2.64990636.268662337.1603553558-977760105.1588733325**
C++	redis-plus-plus	**https://github.com/sewenew/redis-plus-plus?_ga=2.64990636.268662337.1603553558-977760105.1588733325#redis-cluster**
Node.js	node-redis io-redis	**https://github.com/NodeRedis/node-redis** **https://github.com/luin/ioredis**

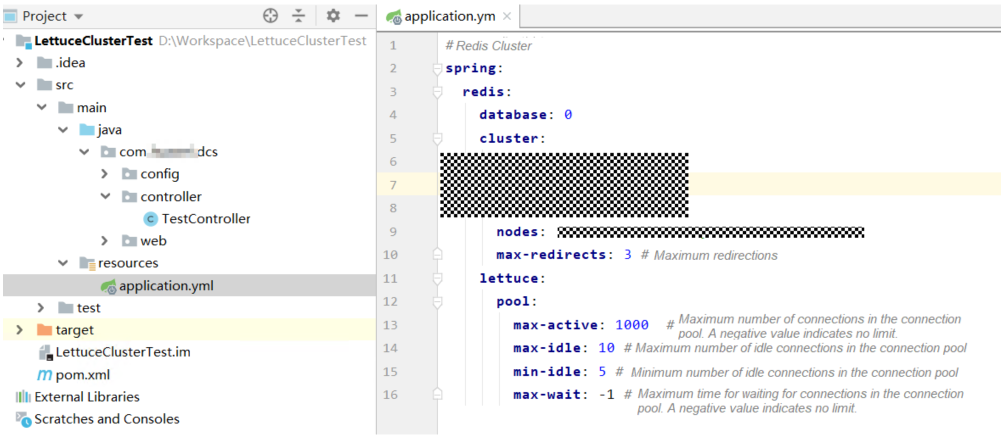
To view all Redis clients, see **https://redis.io/clients**.

14.3.33 Why Do I Need to Configure Timeout for Redis Cluster?

If timeout is not configured, connections will fail.

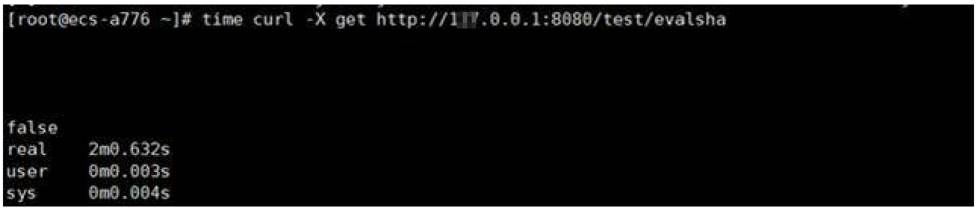
When you connect to a Redis Cluster instance using Spring Boot and Lettuce, connect to all cluster nodes, including faulty replicas.

- If timeout is not configured, minute-level blocking (120s in earlier Lettuce versions and 60s in the new version) may occur when there is a faulty replica, as shown in the following figure.



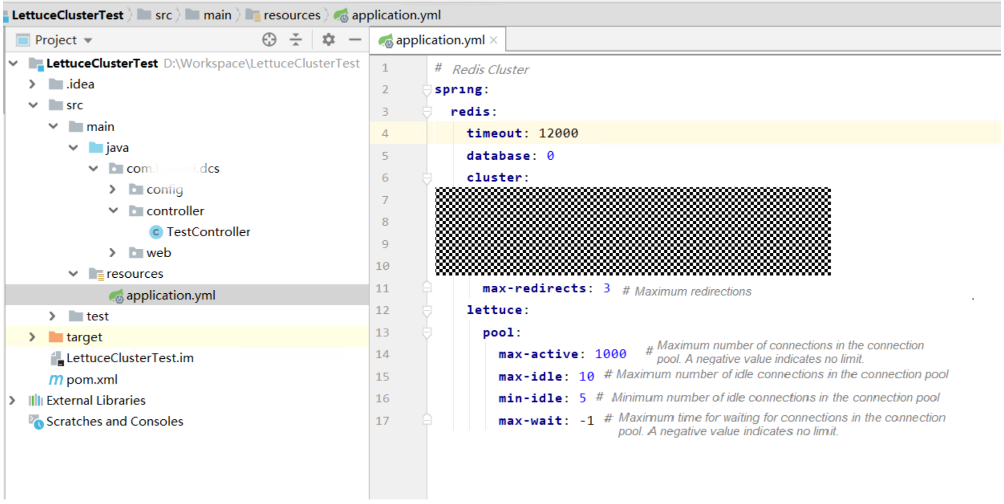
```
# Redis Cluster
spring:
  redis:
    database: 0
    cluster:
[redacted]
      nodes: [redacted]
      max-redirects: 3 # Maximum redirections
    lettuce:
      pool:
        max-active: 1000 # Maximum number of connections in the connection pool. A negative value indicates no limit.
        max-idle: 10 # Maximum number of idle connections in the connection pool
        min-idle: 5 # Minimum number of idle connections in the connection pool
        max-wait: -1 # Maximum time for waiting for connections in the connection pool. A negative value indicates no limit.
```

The end-to-end service access time may reach the maximum timeout, as shown in the following figure.



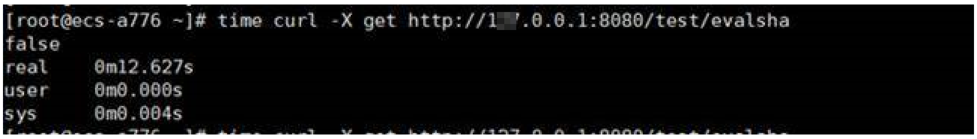
```
[root@ecs-a776 ~]# time curl -X get http://1[redacted].0.0.1:8080/test/evalsha



false
real    2m0.632s
user    0m0.003s
sys     0m0.004s
```

- After the **timeout** parameter is configured on the client, the timeout on the replica will be greatly shortened. You can adjust the timeout based on the service requirements. Assume that the configuration is as follows.



```
#  Redis Cluster
spring:
  redis:
    timeout: 12000
    database: 0
    cluster:
[redacted]
      max-redirects: 3 # Maximum redirections
    lettuce:
      pool:
        max-active: 1000 # Maximum number of connections in the connection pool. A negative value indicates no limit.
        max-idle: 10 # Maximum number of idle connections in the connection pool
        min-idle: 5 # Minimum number of idle connections in the connection pool
        max-wait: -1 # Maximum time for waiting for connections in the connection pool. A negative value indicates no limit.
```

The following figure shows the end-to-end service access time after the configuration is complete.



```
[root@ecs-a776 ~]# time curl -X get http://1[redacted].0.0.1:8080/test/evalsha
false
real    0m12.627s
user    0m0.000s
sys     0m0.004s
```

If the **timeout** parameter is not configured and there is a faulty node, client connections will be blocked.

Configure the timeout based on what the service can tolerate. For example, if you need to request Redis twice in an HTTP request and the maximum timeout of an HTTP request is 10s, it is recommended that you set the timeout in Redis to 5s.

This prevents service interruption if faults occur due to a long timeout duration or no timeout duration.

14.3.34 Can I Change the AZ for an Instance?

No.

To use a different AZ, create another instance in the desired AZ, and then migrate data and switch IP addresses. For details, see **Switching DCS Instance IP Addresses**.

14.3.35 Explaining and Using Hash Tags

Hash Tag Design

Multi-key operations, such as those using the **MSET** command or Lua scripts, are atomic. All specified keys are executed at the same time. However, in a cluster, each key is hashed to a given shard, and multi-key operations are no longer atomic. The keys may be allocated to different slots. As a result, some keys are updated, while others are not. If there is a hash tag, the cluster determines which slot to allocate a key based on the hash tag. Keys with the same hash tag are allocated to the same slot.

Using Hash Tags

Only the content between the first left brace ({) and the following first right brace (}) is hashed.

For example:

- In keys **{user1000}.following** and **{user1000}.followers**, there is only one pair of braces. **user1000** will be hashed.
- In key **foo{}{bar}**, there is no content between the first { and the first }. The whole key **foo{}{bar}** will be hashed as usual.
- In key **foo{{bar}}zap**, **{bar** (the content between the first { and the first }) is hashed.
- In key **foo{bar}{zap}**, **bar** is hashed because it is between the first pair of { and }.

Hash Tag Example

When the following operation is performed:

```
EVAL "redis.call('set',KEYS[1],ARGV[1]) redis.call('set',KEYS[2],ARGV[2])" 2 key1 key2 value1 value2
```

The following error is displayed:

ERR 'key1' and 'key2' not in the same slot

You can use a hash tag to solve this issue:

```
EVAL "redis.call('set',KEYS[1],ARGV[1]) redis.call('set',KEYS[2],ARGV[2])" 2 {user}key1 {user}key2 value1
value2
```

14.3.36 Will Cached Data Be Retained After an Instance Is Restarted?

After a single-node DCS instance is restarted, data in the instance is deleted.

Master/Standby and cluster instances (except single-replica clusters) support AOF persistence by default. Data is retained after these instances are restarted. If AOF persistence is disabled (**appendonly** is set to **no**), data is deleted after the instances are restarted.

14.4 Redis Commands

14.4.1 How Do I Clear Redis Data?

Exercise caution when clearing data.

- Redis 3.0

  Data of a DCS Redis 3.0 instance cannot be cleared on the console, and can only be cleared by the **FLUSHDB** or **FLUSHALL** command in redis-cli.

  Run the **FLUSHALL** command to clear all the data in the instance.

  Run the **FLUSHDB** command to clear the data in the currently selected DB.

- Redis 4.0 and later

  To clear data of a DCS Redis 4.0 and later instance, you can run the **FLUSHDB** or **FLUSHALL** command in redis-cli, use the data clearing function on the DCS console, or run the **FLUSHDB** command on Web CLI.

  To clear data of a Redis Cluster instance, run the **FLUSHDB** or **FLUSHALL** command on every shard of the instance. Otherwise, data may not be completely cleared.

  NOTE

  - Currently, only DCS Redis 4.0 and later instances support data clearing by using the DCS console and by running the **FLUSHDB** command on Web CLI.
  - When you run the **FLUSHDB** command on Web CLI, only one shard is cleared at a time. If there are multiple shards, connect to the master node of each shard and run the **FLUSHDB** command separately.
  - Redis Cluster data cannot be cleared by using Web CLI.

14.4.2 How Do I Find Specified Keys and Traverse All Keys?

Finding Specified Keys

Big key and hot key analysis does not support key searching with specified conditions. To find keys with the specified prefix or suffix, use the **SCAN** command.

For example, to search for keys that contain the letter *a* in a Redis instance, run the following command in redis-cli:

./redis-cli -h *{redis_address}* -p *{port}* [-a *password*] --scan --pattern '*a*'

Traversing All Keys

Do not use the **KEYS** command to traverse all keys of an instance because the **KEYS** command is complex and may block Redis. To traverse all keys in a DCS Redis instance, run the following command in redis-cli:

./redis-cli -h *{redis_address}* -p *{port}* [-a *password*] --scan --pattern '*'

For details about the **SCAN** command, visit the **Redis official website**.

14.4.3 Why is "permission denied" Returned When I Run the KEYS Command in Web CLI?

The **KEYS** command is disabled in Web CLI, but can be run in **redis-cli**.

14.4.4 How Do I Disable High-Risk Commands?

After creating a DCS Redis 4.0 or later instance, you can rename the following critical commands: Currently, you can only rename the **COMMAND**, **KEYS**, **FLUSHDB**, **FLUSHALL**, **HGETALL**, **SCAN**, **HSCAN**, **SSCAN**, and **ZSCAN** commands.

Rename them during instance creation or on the console after the instance is created. To do so, choose **More** > **Command Renaming** in the instance list.

NOTE

- Currently, Redis does not support disabling of commands. To avoid risks when using the preceding commands, rename them. For details about the supported and disabled commands in DCS, see **Command Compatibility**.
- The system will restart the instance after you rename commands. The new commands take effect after the restart.
- Remember the new command names because they will not be displayed on the console for security purposes.

14.4.5 Does DCS for Redis Support Pipelining?

Yes.

For Redis Cluster instances, ensure that all commands in a pipeline are executed on the same shard.

14.4.6 Does DCS for Redis Support the INCR and EXPIRE Commands?

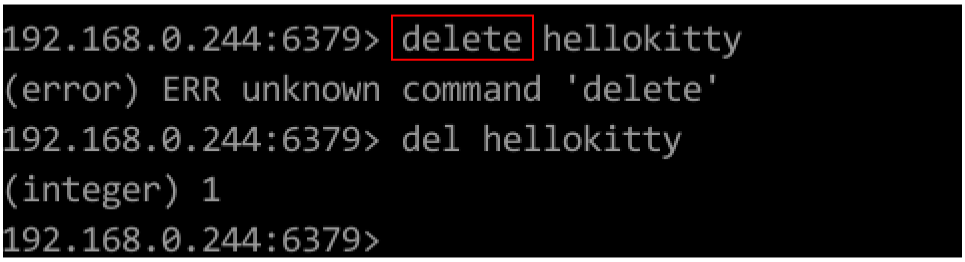
Yes. For more information about Redis command compatibility, see **Command Compatibility**.

14.4.7 Why Do I Fail to Execute Some Redis Commands?

Possible causes include the following:

- The command is incorrect.

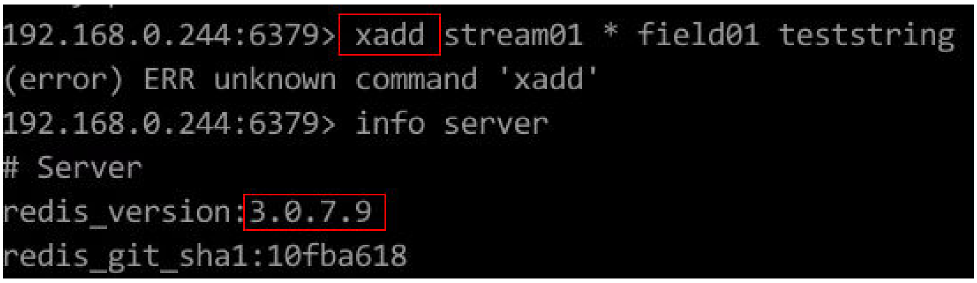
  As shown in the following figure, the error message is returned because the correct command for deleting a key should be **del**.



```
192.168.0.244:6379> delete hellokitty
(error) ERR unknown command 'delete'
192.168.0.244:6379> del hellokitty
(integer) 1
192.168.0.244:6379>
```

- A command available in a higher Redis version is run in a lower Redis version.

  As shown in the following figure, the error message is returned because a stream command (available in Redis 5.0) is run in Redis 3.0.



```
192.168.0.244:6379> xadd stream01 * field01 teststring
(error) ERR unknown command 'xadd'
192.168.0.244:6379> info server
# Server
redis_version:3.0.7.9
redis_git_sha1:10fba618
```

- The command is disabled in DCS.

  For security purposes, some Redis commands are disabled in DCS. For details about disabled and restricted Redis commands, see **Command Compatibility**.

- The LUA script fails to be executed.

  For example, the error message "ERR unknown command 'EVAL'" indicates that your DCS Redis instance is of a lower version that does not support the LUA script. In this case, contact technical support for the instance to be upgraded.

- The **CLIENT SETNAME** and **CLIENT GETNAME** commands fail to be executed.

  This is because the DCS Redis instance is of a lower version that does not support these commands. In this case, contact technical support for the instance to be upgraded.

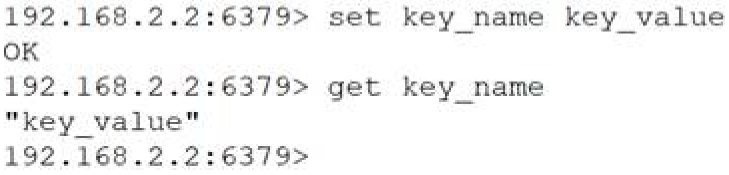
14.4.8 Why Does a Redis Command Fail to Take Effect?

Run the command in redis-cli to check whether the command takes effect.

The following describes two scenarios:

- Scenario 1: Set and query the value of a key to check whether the **SET** and **GET** commands work.

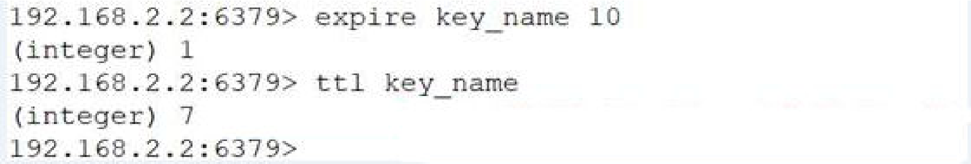
  The **SET** command is used to set the string value. If the value is not changed, run the following commands in redis-cli to access the instance:



```
192.168.2.2:6379> set key_name key_value
OK
192.168.2.2:6379> get key_name
"key_value"
192.168.2.2:6379>
```

- Scenario 2: If the timeout set using the **EXPIRE** command is incorrect, perform the following operations:

Set the timeout to 10 seconds and run the **TTL** command to view the remaining time. As shown in the following example, the remaining time is 7 seconds.



```
192.168.2.2:6379> expire key_name 10
(integer) 1
192.168.2.2:6379> ttl key_name
(integer) 7
192.168.2.2:6379>
```

NOTE

Redis clients (including redis-cli, Jedis clients, and Python clients) communicate with Redis server using a binary protocol.

If Redis commands are run properly in redis-cli, the problem may lie in the service code. In this case, create logs in the code for further analysis.

14.4.9 Is There a Time Limit on Executing Redis Commands? What Will Happen If a Command Times Out?

Redis timeouts happen on the client or server.

- Timeouts on the client are managed in the client code. Services can determine an appropriate timeout. For example, parameter **timeout** can be configured on Java Lettuce.

  When a command times out on the client, errors, command blocks, or client connection retries may occur.

- On the server, the **timeout** parameter is set to **0** by default, indicating that connections will not be terminated. To modify the setting, see **Modifying Configuration Parameters**.

  If the **timeout** parameter is not **0**, idle connections between the client and server will be terminated as specified.

14.4.10 Can I Configure Redis Keys to Be Case-Insensitive?

No. Like in open-source Redis, keys in DCS for Redis are case-sensitive and cannot be configured to be case-insensitive.

14.4.11 Can I View the Most Frequently Used Redis Commands?

No. Redis does not record commands and does not support viewing the most frequently used commands.

14.4.12 Common Web CLI Errors

1. ERR Wrong number of arguments for 'xxx' command

   This error indicates that the executed Redis command has a parameter error (syntax error). Rewrite the command by referring to the open-source Redis command protocol.

2. ERR unknown command 'xxx'

This error indicates that the command is unknown or is not a valid command defined by Redis. Rewrite the command by referring to the open-source Redis command protocol.

3. ERR Unsupported command: 'xxx'

   This error indicates that the command is disabled for DCS Redis instances. For details, see **Web CLI Commands**.

14.5 Instance Scaling and Upgrade

14.5.1 Can DCS Redis Instances Be Upgraded, for Example, from Redis 4.0 to 5.0?

No. Different Redis versions use different underlying architectures. The Redis version used by a DCS instance cannot be changed once the instance is created. For example, you cannot change a DCS Redis 4.0 instance to Redis 5.0. However, you will be informed of any defects or problems found in Redis.

If your service requires the features of higher Redis versions, create a DCS Redis instance of a higher version and then migrate data from the original instance to the new one. For details on how to migrate data, see **Migrating Data with DCS**.

14.5.2 Are Services Interrupted If Maintenance is Performed During the Maintenance Time Window?

O&M personnel will contact you before performing maintenance during the maintenance time window, informing you of the operations and their impacts. You do not need to worry about instance running exceptions.

14.5.3 Are Instances Stopped or Restarted During Specification Modification?

No. Specification modifications can take place while the instance is running and do not affect any other resources.

14.5.4 Are Services Interrupted During Specification Modification?

You are advised to change the instance specifications during off-peak hours because specification modification has the following impacts:

Change of the Instance Type

- **Supported instance type changes:**
  - From single-node to master/standby: Supported by Redis 3.0 and Memcached, and not by Redis 4.0/5.0/6.0.
  - From master/standby to Proxy Cluster: Supported by Redis 3.0, and not by Redis 4.0/5.0/6.0.

    If the data of a master/standby DCS Redis 3.0 instance is stored in multiple databases, or in non-DB0 databases, the instance cannot be changed to the Proxy Cluster type. A master/standby instance can be changed to the Proxy Cluster type only if its data is stored only on DB0.

- From cluster types to other types: Not supported.

- **Impact of instance type changes:**
  - From single-node to master/standby for a DCS Redis 3.0 instance:

    The instance cannot be connected for several seconds and remains read-only for about 1 minute.
  - From master/standby to Proxy Cluster for a DCS Redis 3.0 instance:

    The instance cannot be connected and remains read-only for 5 to 30 minutes.

Scaling

- **The following table lists scaling options supported by different DCS instances.**

Table 14-9 Scaling options supported by different DCS instances

Cache Engine	Single-Node	Master/Standby	Redis Cluster	Proxy Cluster
Redis 3.0	Scaling up/down	Scaling up/down	N/A	Scaling out
Redis 4.0	Scaling up/down	Scaling up/down and replica quantity change	Scaling up/out, down/in, and replica quantity change	N/A
Redis 5.0	Scaling up/down	Scaling up/down and replica quantity change	Scaling up/out, down/in, and replica quantity change	N/A
Redis 6.0	Scaling up/down	Scaling up/down	N/A	N/A
Memcached	Scaling up/down	Scaling up/down	N/A	N/A

NOTE

If the reserved memory of a DCS Redis 3.0 or Memcached instance is insufficient, the scaling may fail when the memory is used up.

Change the replica quantity and capacity separately.

- **Impact of scaling:**
  - Single-node and master/standby

    - A DCS Redis 4.0/5.0/6.0 instance will be disconnected for several seconds and remain read-only for about 1 minute.
    - A DCS Redis 3.0 instance will be disconnected and remain read-only for 5 to 30 minutes.
    - For scaling up, only the memory of the instance is expanded. The CPU processing capability is not improved.
    - Data of single-node instances may be lost because they do not support data persistence. After scaling, check whether the data is complete and import data if required.
    - Backup records of master/standby instances cannot be used after scaling down.
  - Cluster
    - If the shard quantity is not decreased, the instance can always be connected, but the CPU usage will increase, compromising performance by up to 20%, and the latency will increase during data migration.
    - During scaling up, new Redis Server nodes are added, and data is automatically balanced to the new nodes.
    - Nodes will be deleted if the shard quantity decreases. To prevent disconnection, ensure that the deleted nodes are not directly referenced in your application.
    - Ensure that the used memory of each node is less than 70% of the maximum memory per node of the new flavor. Otherwise, you cannot perform the scale-in.
    - If the memory becomes full during scaling due to a large amount of data being written, scaling will fail. Modify specifications during off-peak hours.
    - Scaling involves data migration. The latency for accessing the key being migrated increases. For a Redis Cluster instance, ensure that the client can properly process the **MOVED** and **ASK** commands. Otherwise, requests will fail.
    - Before scaling, perform cache analysis to ensure that no big keys (≥ 512 MB) exist in the instance. Otherwise, scaling may fail.
    - Backup records created before scaling cannot be restored.

- **Notes on changing the number of replicas of a DCS Redis instance:**

  Deleting replicas interrupts connections. If your application cannot reconnect to Redis or handle exceptions, you need to restart the application after scaling.

14.5.5 Why Can't I Modify Specifications for a DCS Redis/ Memcached Instance?

- Check whether other tasks are running.

  Specifications of a DCS instance cannot be modified if another task of the instance is still running. For example, you cannot delete or scale up an instance while it is being restarted. Likewise, you cannot delete an instance while it is being scaled up.

  If the specification modification fails, try again later. If it fails again, contact technical support.

- Modify instance specifications during off-peak hours. If the modification failed in peak hours (for example, when memory or CPU usage is over 90% or write traffic surges), try again during off-peak hours.

14.5.6 How Do I Reduce the Capacity of a DCS Instance?

The following table lists scaling options supported by different DCS instances.

Table 14-10 Scaling options supported by different DCS instances

Cache Engine	Single-Node	Master/ Standby	Redis Cluster	Proxy Cluster
Redis 3.0	Scaling up/ down	Scaling up/ down	N/A	Scaling out
Redis 4.0	Scaling up/ down	Scaling up/ down and replica quantity change	Scaling up/ out, down/in, and replica quantity change	N/A
Redis 5.0	Scaling up/ down	Scaling up/ down and replica quantity change	Scaling up/ out, down/in, and replica quantity change	N/A
Redis 6.0	Scaling up/ down	Scaling up/ down	N/A	N/A
Memcached	Scaling up/ down	Scaling up/ down	N/A	N/A

For details about how to change the capacity, see **Modifying DCS Instance Specifications**.

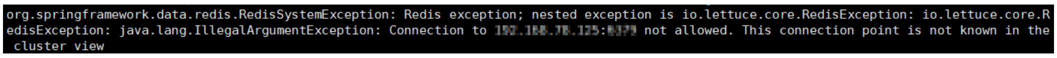
If you want to use a smaller Proxy Cluster DCS Redis 3.0 instance, back up the data of the existing instance, and create a new Proxy Cluster instance with the desired capacity. Then, import the backup data to the new instance. After the data migration is complete, delete the old instance. For details about data migration operations, see **Importing Backup Files**.

14.5.7 How Do I Handle an Error When I Use Lettuce to Connect to a Redis Cluster Instance After Specification Modification?

Symptom

If the shard quantity changes during specification modification of a Redis Cluster instance, some slots are migrated to new shards. The following error occurs when you use Lettuce to connect to the instance.

Figure 14-7 Error



```
org.springframework.data.redis.RedisSystemException: Redis exception; nested exception is io.lettuce.core.RedisException: io.lettuce.core.R
edisException: java.lang.IllegalArgumentException: Connection to [redacted]:[redacted] not allowed. This connection point is not known in the
 cluster view
```

For details, see **Connection to X not allowed. This connection point is not known in the cluster view**.

Analysis

Specification modification process of a Redis Cluster instance:

After being started, the client obtains the cluster node topology by using the **Cluster Nodes** command based on RESP2, and maintains the topology in its in-memory data structure.

For data access, the client uses the CRC16 algorithm to calculate the hash slot of a key, and automatically routes requests based on the topology and slot information stored in the memory.

If the number of shards changes during scaling, the topology and slot mapping changes. In this case, the client needs to automatically update the topology. Otherwise, the request route may fail or the route location may be incorrect. As a result, an error is reported during client connection.

For example, when the number of shards in a Redis Cluster instance changes from three to six, the topology and slot mapping changes as shown in the following figures.

Figure 14-8 A Redis Cluster instance before scaling up

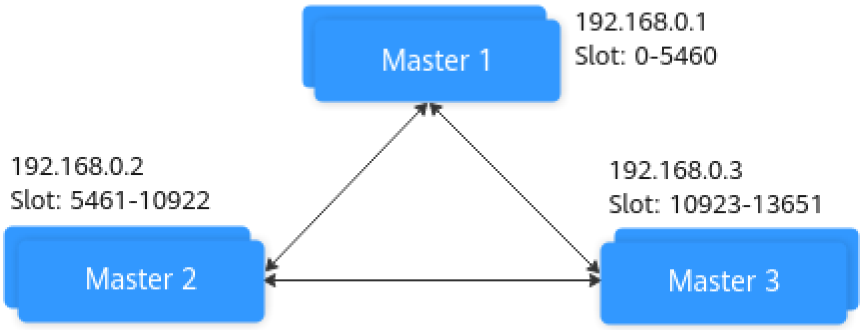
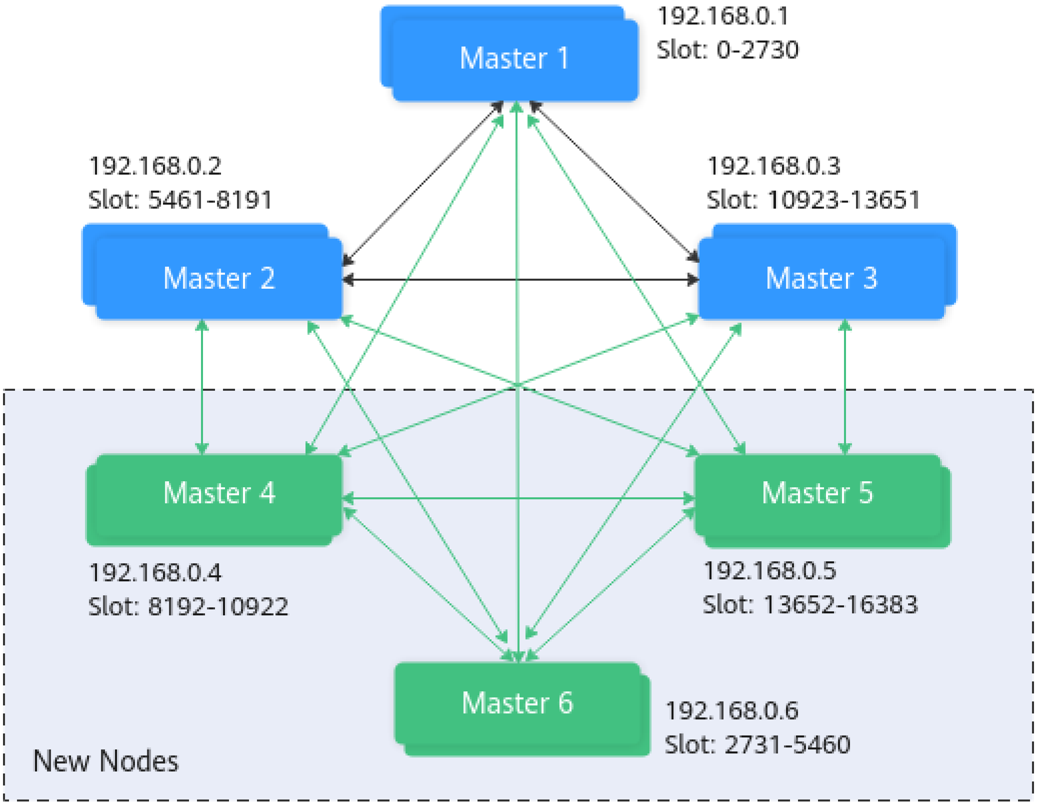


Figure 14-9 A Redis Cluster instance after scaling up



Solutions

Solution 1 (Recommended)

Enable automated topology refresh.

```
ClusterTopologyRefreshOptions topologyRefreshOptions = ClusterTopologyRefreshOptions.builder()
            // Periodic refresh: every time milliseconds.
           .enablePeriodicRefresh(Duration.ofMillis(time))
        // Triggers of adaptive refresh: MOVED redirection, ASK redirection, reconnection, unknown node
(since 5.1), and slot not in any of the current shards (since 5.2).
           .enableAllAdaptiveRefreshTriggers()
           .build();
```

For details, see **an example of using Lettuce to connect to a Redis Cluster instance**.

NOTE

If you use Lettuce to connect to a Redis Cluster instance and automated refresh is not enabled, you need to restart the client after specification modification.

Solution 2

Disable validation of cluster node membership.

```
ClusterClientOptions clusterClientOptions = ClusterClientOptions.builder()
       .validateClusterNodeMembership(false)
       .build();
```

If **validateClusterNodeMembership** is **true**, check whether the current connection address is in the cluster topology obtained through **CLUSTER NODES**, before connecting to the cluster. If it is not in the topology, the error occurs.

> **NOTE**
>
> Impact of disabling validation of cluster node membership:
>
> - Lack of security breach detection.
> - If automated topology refresh is disabled, a MOVED redirection request may be generated after the Redis Cluster specifications are changed and the shard quantity increases. Redirection increases the network load of the cluster and the time required to process a single request. If the shard quantity decreases, deleted shards cannot be connected.

14.6 Monitoring and Alarm

14.6.1 How Do I View Current Concurrent Connections and Maximum Connections of a DCS Redis Instance?

Viewing Concurrent Connections of a DCS Redis Instance

The number of connections received by a DCS instance is a metric that is monitored by Cloud Eye. For details on how to view metrics, see **Viewing DCS Monitoring Metrics**.

On the Cloud Eye console, find the **Connected Clients** metric. Click ⤢ to view monitoring details on an enlarged graph.

Specify a time range to view the metric in a specific monitoring period. For example, you can select a 10-minute period to view the number of connections received during the period. On the graph, you can view the trend and the total number of connections received during the period.

On the Cloud Eye console, you can also view other monitoring metrics of your DCS instances, for example:

- CPU Usage
- Memory Usage
- Used Memory
- Ops per Second

Viewing or Modifying the Maximum Connections of an Instance

When creating an instance on the console, you can view the default maximum number of connections and the limit that can be configured.

After an instance is created, you can view or change the value of **maxclients** (the maximum number of connections) on the **Instance Configuration** > **Parameters** page of the DCS console. (This parameter cannot be modified for Proxy Cluster instances.)

14.6.2 Does Redis Support Command Audits?

No. To ensure high-performance reads and writes, Redis does not audit commands. Commands are not printed.

14.6.3 What Should I Do If the Monitoring Data of a DCS Redis Instance Is Abnormal?

If you have any doubt on the monitoring data of a DCS Redis instance, you can access the instance through redis-cli and run the **INFO ALL** command to view the metrics. For details about the output of the **INFO ALL** command, see **https://redis.io/docs/latest/commands/info/**.

14.6.4 Why Is Available Memory of Unused DCS Instances Less Than Total Memory and Why Is Memory Usage of Unused DCS Instances Greater Than Zero?

For DCS Redis 3.0 instances and Memcached instances, the available memory is less than the total memory because some memory is reserved for system overhead and data persistence (supported by master/standby instances). DCS instances use a certain amount of memory for Redis-server buffers and internal data structures. This is why memory usage of unused DCS instances is greater than zero.

14.6.5 Why Is Used Memory Greater Than Available Memory?

For single-node and master/standby DCS instances, the used instance memory is measured by the Redis-server process. For cluster DCS instances, the used cluster memory is the sum of used memory of all shards in the cluster.

Due to the internal implementation of the open-source redis-server, the used instance memory is normally slightly higher than the available instance memory.

Why is used_memory higher than max_memory?

Redis allocates memory using zmalloc. It does not check whether used_memory exceeds max_memory every time the memory is allocated. Instead, it checks whether the current used_memory exceeds max_memory at the beginning of a periodic task or command processing. If used_memory exceeds max_memory, eviction is triggered. Therefore, the restrictions of the max_memory policy are not implemented in real time or rigidly. A case in which the used_memory is greater than the max_memory may occur occasionally.

14.6.6 Why Does Bandwidth Usage Exceed 100%?

The basic information about the bandwidth usage metric is as follows.

Metric ID	Metric Name	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
bandwidth_usage	Bandwidth Usage	Percentage of the used bandwidth to the maximum bandwidth limit	0–200%	Monitored object: Redis 4.0 and later Redis Server of a master/standby or cluster instance Dimension: dcs_cluster_node	1 minute

Bandwidth usage = (Input flow + Output flow)/(2 x Maximum bandwidth) x 100%

According to the formula, the bandwidth usage counts in the input flow and output flow, which include the traffic for replication between the master and replicas. Therefore, the total bandwidth usage is larger than the normal service traffic, and may exceed 100%.

If the value of the **Flow Control Times** metric is larger than 0, the maximum bandwidth has been reached and flow control has been performed.

However, flow control decisions are made without considering the traffic for replication between the master and replicas. Therefore, sometimes the bandwidth usage exceeds 100% but the number of flow control times is 0.

14.6.7 Why Is the Rejected Connections Metric Displayed?

If the **Rejected Connections** metric is displayed, check if the number of connected clients exceeds the maximum allowed number of connections of the instances.

- To check the maximum allowed number of connections, go to the **Parameters** tab page of the instance and check the value of the **maxclients** parameter.
- To check the current number of connections, go to the **Performance Monitoring** tab page of the instance and check the **Connected Clients** metric.

If the current number of connections reaches the upper limit, you can adjust the value of **maxclients**. If the value of **maxclients** can no longer be increased, increase the instance specifications.

14.6.8 Why Is Flow Control Triggered? How Do I Handle It?

Flow control is triggered when the traffic used by a Redis instance in a period exceeds the maximum bandwidth.

NOTE

For details about the maximum bandwidth of an instance flavor, see **Instance Specifications**, or refer to the page of creating an instance on the console.

Even if the bandwidth usage is low, flow control may still be triggered. The real-time bandwidth usage is reported once in each reporting period. Flow controls are checked every second. The traffic may surge within seconds and then fall back between reporting periods. By the time the bandwidth usage is reported, it may have already restored to the normal level.

For master/standby instances:

- If flow control is always triggered when the bandwidth usage is low, there may be service microbursts or big or hot keys. In this case, check for big or hot keys.
- If the bandwidth usage remains high, the bandwidth limit may be exceeded. In this case, expand the capacity. Larger capacity supports higher bandwidth.

For cluster instances:

- If flow control is triggered only on one or a few shards, the shards may have big or hot keys.
- If flow control or high bandwidth usage occurs on all or most shards at the same time, bandwidth usage of the instance has reached the limit. In this case, expand the instance capacity.

NOTE

- You can analyze big keys and hot keys on the DCS console. For details, see **Analyzing Big Keys and Hot Keys**.
- Running commands (such as **KEYS**) that consume lots of resources may cause high CPU and bandwidth usage. As a result, flow control is triggered.

14.7 Data Backup, Export, and Migration

14.7.1 How Do I Export DCS Redis Instance Data?

- For master/standby or cluster instances:

  Perform the following operations to export the data:

  a. On the **Backups & Restorations** page, view the backup records.

  b. If there are no backup records, create a backup manually and download the backup file as prompted.

  NOTE

  If your DCS instances were created a long time ago, the versions of these instances may not be advanced enough to support some new functions (such as backup and restoration). You can contact technical support to upgrade your DCS instances. After the upgrade, you can back up and restore your instances.

- For single-node instances:

  Single-node instances do not support the backup function. You can use redis-cli to export RDB files. This operation depends on **SYNC** command.

- If the instance allows the **SYNC** command (such as a Redis 3.0 single-node instance), run the following command to export the instance data:

  redis-cli -h {source_redis_address} -p 6379 [-a password] --rdb {output.rdb}

- If the instance does not allow the **SYNC** command (such as a Redis 4.0 or 5.0 single-node instance), migrate the instance data to a master/standby instance and export the data by using the backup function.

14.7.2 Can I Export Backup Data of DCS Redis Instances to RDB Files Using the Console?

- Redis 3.0

  No. On the console, backup data of a DCS Redis 3.0 instance can be exported only to AOF files. To export data to RDB files, run the following command in redis-cli:

  redis-cli -h *{redis_address}* -p 6379 [-a *password*] --rdb {output.rdb}

- Redis 4.0 and later

  Yes. DCS Redis 4.0 and later instances support AOF and RDB persistence. You can back up data to RDB and AOF files on the console and download the files.

14.7.3 Why Are Processes Frequently Killed During Data Migration?

Possible cause: The memory is insufficient.

Solution: Expand the memory of the server on which the migration command is executed.

14.7.4 Is All Data in a DCS Redis Instance Migrated During Online Migration?

Migration between single-node and master/standby instances involves the full set of data. After the migration, a given key will remain in the same DB as it was before the migration.

By contrast, a cluster instance only has one DB, which is DB0. During the migration, data in all slots of DB0 is migrated.

14.7.5 Do DCS Redis Instances Support Data Persistence? What Is the Impact of Persistence?

Is Persistence Supported?

Single-node: No

Master/Standby and cluster (except single-replica clusters): Yes

How Is Data Persisted?

- DCS Redis instances supports only AOF persistence by default. You can enable or disable persistence as required. All instances except single-node and single-replica cluster ones are created with AOF persistence enabled.
- DCS Redis instances do not support RDB persistence by default and the **save** parameter cannot be manually configured. If RDB persistence is required for a master/standby or cluster instance of Redis 4.0 or later, you can use the backup and restoration function to back up the instance data in an RDB file and store the data in OBS.

Disk Used for Persistence

For DCS Redis 4.0 and later instances, data is persisted to SSD disks.

Impact of AOF Persistence

After AOF persistence is enabled, the Redis-Server process records operations in the AOF file for data persistence. This may have the following impacts:

- If the disk or I/O of the underlying compute node is faulty, the latency may increase or a master/standby switchover may occur.
- Redis-Server periodically rewrites the AOF. During a rewrite, the latency may be high for a short time. For details about the AOF rewriting rules, see **When Will AOF Rewrites Be Triggered?**

If DCS instances are used for application acceleration, you are advised to disable AOF persistence for higher performance and stability.

Exercise caution when disabling AOF persistence. After it is disabled, cached data may be lost in extreme scenarios, for example, when both the master and standby nodes are faulty.

To disable AOF persistence, set parameter **appendonly** to **no** on the instance details page.

14.7.6 When Will AOF Rewrites Be Triggered?

AOF rewrites involve the following concepts:

- Rewrite window, which is currently 01:00 to 04:59
- Disk usage threshold, which is 50%
- Dataset memory, which is the percentage of memory that Redis dataset has used.

AOF rewrites are triggered in the following scenarios:

- If the disk usage reaches the threshold (regardless of whether the current time is within the rewrite window), rewrites will be triggered on instances whose AOF file size is larger than the dataset memory.
- If the disk usage is below the threshold and the current time is within the rewrite window, rewrites will be triggered on instances whose AOF file size is larger than the dataset memory multiplied by 1.5.

- If the disk usage is below the threshold but the current time is out of the rewrite window, rewrites will be triggered on instances whose AOF file size is larger than the instance's maximum memory multiplied by 4.5.

14.7.7 Can I Migrate Data to Multiple Target Instances in One Migration Task?

No. A migration task allows data to be migrated to only one target instance. To migrate data to multiple target instances, create multiple migration tasks.

14.7.8 How Do I Enable the SYNC and PSYNC Commands?

- Migration within DCS:
  - By default, the **SYNC** and **PSYNC** commands can be used when self-hosted Redis is migrated to DCS.
  - During online migration between DCS Redis instances in the same region under the same account, the **SYNC** and **PSYNC** commands are automatically enabled.
  - During online migration between DCS Redis instances in different regions or under different accounts within a region, the **SYNC** and **PSYNC** commands are not automatically enabled, and online migration cannot be used. You can migrate data using backup files.
- Migration from other cloud vendors to DCS:
  - Generally, cloud vendors disable the **SYNC** and **PSYNC** commands. If you want to use the online migration function on the DCS console, contact the O&M personnel of the source cloud vendor to enable the commands. For offline migration, you can import backup files.
  - If incremental migration is not required, you can perform full migration by referring to **Online Full Migration of Redis from Another Cloud with redis-shake**. This method does not depend on **SYNC** and **PSYNC**.

14.7.9 Will the Same Keys Be Overwritten During Data Migration or Backup Import?

If the data exists on both the source and target instances, the target data is overwritten by the source data. If the data exists only on the target instance, the data will be retained.

Inconsistency between source and target data after the migration may be due to the target data that existed and was retained before migration.

14.7.10 Online Migration with Rump

Background

Rump is an open-source tool designed for migrating Redis data online. It supports migration between DBs of the same instance and between DBs of different instances.

Migration Principles

Rump uses the **SCAN** command to acquire keys and the **DUMP**/**RESTORE** command to get or set values.

Featuring time complexity O(1), **SCAN** is capable of quickly getting all keys. **DUMP**/**RESTORE** is used to read/write values independent from the key type.

Rump brings the following benefits:

- The **SCAN** command replaces the **KEYS** command to avoid blocking Redis.
- Any type of data can be migrated.
- **SCAN** and **DUMP**/**RESTORE** operations are pipelined, improving the network efficiency during data migration.
- No temporary file is involved, saving disk space.
- Buffered channels are used to optimize performance of the source server.

NOTICE

1. To cluster DCS instances, you cannot use Rump. Instead, use redis-port or redis-cli.
2. To prevent migration command resolution errors, do not include special characters (#@:) in the instance password.
3. Stop the service before migrating data. If data is kept being written in during the migration, some keys might be lost.

Step 1: Installing Rump

1. Download **Rump (release version)**.

   On 64-bit Linux, run the following command:

   wget https://github.com/stickermule/rump/releases/download/0.0.3/rump-0.0.3-linux-amd64;

2. After decompression, run the following commands to add the execution permission:

   mv rump-0.0.3-linux-amd64 rump;

   chmod +x rump;

Step 2: Migrating Data

rump -from {*source_redis_address*} **-to** {*target_redis_address*}

Parameter/Option description:

- *{source_redis_address}*

  Source Redis instance address, in the format of redis://[user:password@]host:port/db. **[user:password@]** is optional. If the instance is accessed in password-protected mode, you must specify the password in the RFC 3986 format. **user** can be omitted, but the colon (:) cannot be omitted. For example, the address may be **redis://:mypassword@192.168.0.45:6379/1**.

db is the sequence number of the database. If it is not specified, the default value is 0.

- {*target_redis_address*}

  Address of the target Redis instance, in the same format as the source.

  In the following example, data in DB0 of the source Redis is migrated to the target Redis whose connection address is 192.168.0.153. ********** stands for the password.

```
[root@ecs ~]# ./rump -from redis://127.0.0.1:6379/0  -to redis://:******@192.168.0.153:6379/0
.Sync done.
[root@ecs ~]#
```

14.7.11 What Should I Consider When Transferring or Operating Data Between Different OSs?

Convert the format of a data file before importing the file.

Run the following command to convert the format of a file in the Windows OS to that in the Unix-like OS:

dos2unix ***{filename}***

Run the following command to convert the format of a file in the Unix-like OS to that in the Windows OS:

unix2dos ***{filename}***

14.7.12 Can I Migrate Data from a Multi-DB Source Redis Instance to a Cluster DCS Redis Instance?

A total of 256 DBs (DB 0 to DB 255) can be configured for a single-node or master/standby DCS instance.

- If the target is a cluster instance (a cluster instance has only one DB):

  Solutions:

  a. Combine different DBs in the source Redis instance into one DB.

  b. Apply for multiple DCS instances.

  After the migration, the instance connection address and DB IDs change. In this case, modify the configurations for your services.

14.7.13 How Can I Migrate Partial Data?

The online migration function provided on the console does not support migration of specified DBs. If you want to migrate **specified Redis DBs**, use redis-shake to export or import the specified DBs.

For details about how to install and use redis-shake, see **Self-Hosted Redis Cluster Migration with redis-shake** and **redis-shake configuration instructions**.

To migrate **specified data**, develop a script to obtain the specified keys and data and then import the data to a DCS instance.

14.7.14 What Are the Constraints and Precautions for Migrating Redis Data to a Cluster Instance?

- Proxy Cluster instances

  Proxy Cluster instances are used in the same way that you use single-node or master/standby instances. However, only one DB is configured for a Proxy Cluster instance by default, and the **SELECT** command is not supported. When data files are imported in batches, an error message will be displayed and ignored if the **SELECT** command exists. Then, the remaining data will continue to be imported.

  Example:

  DB 0 and DB 2 in the source Redis instance contain data, and the generated AOF or RDB file contains these two DBs.

  When the source Redis data is imported into a Proxy Cluster DCS instance, the **SELECT 2** command will be ignored, and then data in DB 2 in the source Redis instance will be imported.

  Note that:

  – If different DBs in the source Redis instance contain the same keys, values of keys in the DB with the largest ID will overwrite those in the other DBs.

  – If the source Redis instance contains multiple DBs, data is stored in the same DB after being migrated to a cluster DCS instance, and the **SELECT** command is not supported. In this case, modify the configurations for your services.

- Redis Cluster instances

  Only one DB is configured for a Redis Cluster instance. Data is migrated to a Redis Cluster instance in a different way from other types of instances. Nodes in the shards of a Redis Cluster must be connected separately through clients. Data is imported to the nodes separately. Run the following command to query the IP addresses of the cluster nodes:

  redis-cli -h *{Redis Cluster IP}* -p 6379 -a *{password}* cluster nodes

  In the returned list of IP addresses, record the ones marked by "master".

14.7.15 What Should I Consider for Online Migration?

- Network

  Before online migration, ensure that the network configured for the migration task is connected to source and target Redis instances.

- Tool

  Use the online migration function provided on the DCS console.

- Data integrity

  If you suspend your services for data migration, check the data volume and main keys after the migration.

  If you do not suspend your services, migrate data incrementally.

- Impact of capacity expansion of the source instance

  During online migration, expanding the source instance's capacity may affect the migration or customer data. If the memory of the source instance becomes insufficient during the migration, stop the migration task and then expand the capacity.

- Timing

  Migration should take place during off-peak hours.

- Version restrictions

  You can migrate data from an earlier version to a later version, and vice versa, but you need to check whether the target instance supports the commands used in your service systems.

14.7.16 Can I Perform Online Migration Without Any Service Interruption?

Yes. You can use the application dual-write mode. In this mode, during data migration, data is still read from the source Redis instance, and operations such as adding, deleting, and modifying data are also performed on the DCS Redis instance.

After maintaining the preceding mode for a period of time (waiting for a large amount of data to be deleted after expiration), migrate the cached data from your service systems to DCS. If service system migration to the cloud service is also involved, deploy your service systems before migrating your cached data.

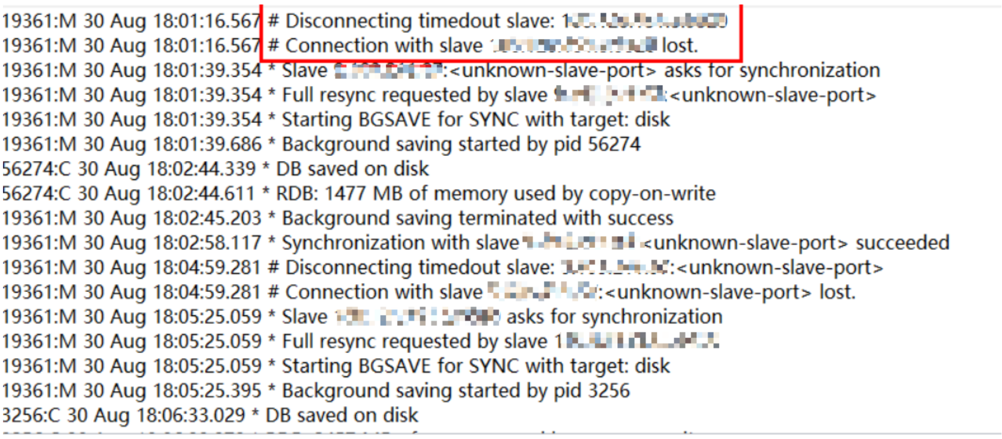
This mode is not recommended for the following reasons:

1. Stable and quick network access cannot be ensured. If the source Redis instance is not deployed on DCS, access DCS over a public network, which is inefficient.
2. Modify the code to implement concurrent writing of two sets of data.
3. The data eviction policy varies depending on the source Redis instance. It may take a long time to complete data migration and it is difficult to ensure data integrity.

14.7.17 What If "Disconnecting timedout slave" and "overcoming of output buffer limits" Are Reported on the Source Instance During Online Migration?

The following error messages may be displayed during online migration:

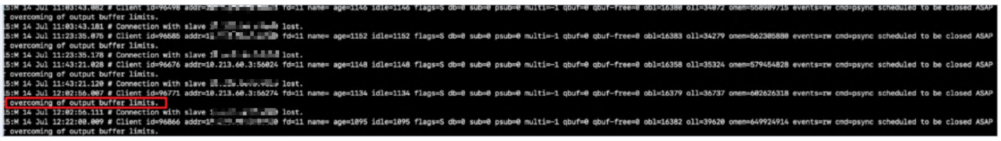
- "Disconnecting timedout slave" is reported on the source instance, as shown in the following figure:



```
19361:M 30 Aug 18:01:16.567 # Disconnecting timedout slave: [redacted]
19361:M 30 Aug 18:01:16.567 # Connection with slave [redacted] lost.
19361:M 30 Aug 18:01:39.354 * Slave [redacted]:<unknown-slave-port> asks for synchronization
19361:M 30 Aug 18:01:39.354 * Full resync requested by slave [redacted]:<unknown-slave-port>
19361:M 30 Aug 18:01:39.354 * Starting BGSAVE for SYNC with target: disk
19361:M 30 Aug 18:01:39.686 * Background saving started by pid 56274
56274:C 30 Aug 18:02:44.339 * DB saved on disk
56274:C 30 Aug 18:02:44.611 * RDB: 1477 MB of memory used by copy-on-write
19361:M 30 Aug 18:02:45.203 * Background saving terminated with success
19361:M 30 Aug 18:02:58.117 * Synchronization with slave [redacted]:<unknown-slave-port> succeeded
19361:M 30 Aug 18:04:59.281 # Disconnecting timedout slave: [redacted]:<unknown-slave-port>
19361:M 30 Aug 18:04:59.281 # Connection with slave [redacted]:<unknown-slave-port> lost.
19361:M 30 Aug 18:05:25.059 * Slave [redacted] asks for synchronization
19361:M 30 Aug 18:05:25.059 * Full resync requested by slave [redacted]
19361:M 30 Aug 18:05:25.059 * Starting BGSAVE for SYNC with target: disk
19361:M 30 Aug 18:05:25.395 * Background saving started by pid 3256
3256:C 30 Aug 18:06:33.029 * DB saved on disk
```

Solution: Set the **repl-timeout** parameter of the source Redis instance to 300s.

- "overcoming of output buffer limits" is reported on the source instance, as shown in the following figure:



```
5:M 14 Jul 11:03:43.082 # Client id=96498 addr=[redacted] fd=11 name= age=1146 idle=1146 flags=S db=0 sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=0 obl=16380 oll=34072 omem=558909715 events=rw cmd=psync scheduled to be closed ASAP
 overcoming of output buffer limits.
5:M 14 Jul 11:03:43.181 # Connection with slave [redacted] lost.
5:M 14 Jul 11:23:35.075 # Client id=96585 addr=[redacted] fd=11 name= age=1152 idle=1152 flags=S db=0 sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=0 obl=16383 oll=34279 omem=562305880 events=rw cmd=psync scheduled to be closed ASAP
 overcoming of output buffer limits.
5:M 14 Jul 11:23:35.178 # Connection with slave [redacted] lost.
5:M 14 Jul 11:43:21.028 # Client id=96676 addr=10.213.60.3:56024 fd=11 name= age=1148 idle=1148 flags=S db=0 sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=0 obl=16358 oll=35324 omem=579454828 events=rw cmd=psync scheduled to be closed ASAP
 overcoming of output buffer limits.
5:M 14 Jul 11:43:21.120 # Connection with slave [redacted] lost.
5:M 14 Jul 12:02:56.007 # Client id=96771 addr=10.213.60.3:56274 fd=11 name= age=1134 idle=1134 flags=S db=0 sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=0 obl=16379 oll=36737 omem=602626318 events=rw cmd=psync scheduled to be closed ASAP
 overcoming of output buffer limits.
5:M 14 Jul 12:02:56.111 # Connection with slave [redacted] lost.
5:M 14 Jul 12:22:00.009 # Client id=96866 addr=[redacted] fd=11 name= age=1095 idle=1095 flags=S db=0 sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=0 obl=16382 oll=39620 omem=649924914 events=rw cmd=psync scheduled to be closed ASAP
 overcoming of output buffer limits.
```

Solution: Set the **client-output-buffer-limit** parameter of the source Redis instance to 20% of the maximum memory of the instance.

14.7.18 Why Is Memory of a DCS Redis Instance Unchanged After Data Migration Using Rump, Even If No Error Message Is Returned?

For details on how to use Rump, see the **Data Migration Guide**.

Possible causes:

- Rump does not support migration to cluster DCS instances.
- Commands are incorrectly run in Rump.

14.7.19 Can I Migrate Data from a Lower Redis Version to a Higher One?

Yes. Redis is backward compatible.

The version of the source Redis (DCS, self-built, or another cloud) can be earlier than or the same as the target DCS instance.

14.7.20 How Do I Migrate Memcached Data?

Memcached does not provide commands for traversal data query. Therefore, you cannot directly export data from your Memcached and migrate the data to a DCS Memcached instance.

Record cache keys through logging of your application systems, extract key-value data, and write the data to a DCS Memcached instance, achieving gradual data migration.

NOTE

By using some open-source tools, you can run the **stats cachedump** command of Memcached and perform get operations to query partial key-value data stored in your Memcached. However, you can only query key data not greater than 2 MB (including the size of all keys queried and additional information with a size of more than 20 bytes for each key) by using this command. Therefore, you cannot use such tools or similar methods for data migration.

14.8 Big/Hot Key Analysis

14.8.1 What Are Big Keys and Hot Keys?

Term	Definition
Big key	There are two types of big keys: ● Keys that have a large value. If the size of a single String key exceeds 10 KB, or if the size of all elements of a key combined exceeds 50 MB, the key is defined as a big key. ● Keys that have a large number of elements. If the number of elements in a key exceeds 5000, the key is defined as a big key.
Hot key	A key is defined as a hot key if it is frequently requested or if it occupies a large number of resources. For example: ● In a cluster instance, a shard processes 10,000 requests per second, among which 3000 are performed on the same key. ● In a cluster instance, a shard uses a total of 100 Mbits/s inbound and outbound bandwidth, among which 80 Mbits/s is used by the **HGETALL** operation on a Hash key.

14.8.2 What Is the Impact of Big Keys or Hot Keys?

Category	Impact
Big key	**Instance specifications fail to be modified.** Specification modification of a Redis Cluster instance involves rebalancing (data migration between nodes). Redis has a limit on key migration. If the instance has any single key bigger than 512 MB, the modification will fail when big key migration between nodes times out. The bigger the key, the more likely the migration will fail.
	Data migration fails. During data migration, if a key has many elements, other keys will be blocked and will be stored in the memory buffer of the migration ECS. If they are blocked for a long time, the migration will fail.

Category	Impact
	Cluster shards are unbalanced. ● The memory usage of shards is unbalanced. For example, if a shard uses a large memory or even uses up the memory, keys on this shard are evicted, and resources of other shards are wasted. ● The bandwidth usage of shards is unbalanced. For example, flow control is repeatedly triggered on a shard.
	Latency of client command execution increases. Slow operations on a big key block other commands, resulting in a large number of slow queries.
	Flow control is triggered on the instance. Frequently reading data from big keys exhausts the outbound bandwidth of the instance, triggering flow control. As a result, a large number of commands time out or slow queries occur, affecting services.
	Master/standby switchover is triggered. If the high-risk **DEL** operation is performed on a big key, the master node may be blocked for a long time, causing a master/standby switchover.
Hot key	**Cluster shards are unbalanced.** If only the shard where the hot key is located is busy processing service queries, there may be performance bottlenecks on a single shard, and the compute resources of other shards may be wasted.
	CPU usage surges. A large number of operations on hot keys may cause high CPU usage. If the operations are on a single cluster shard, the CPU usage of the shard where the hot key is located will surge. This will slow down other requests and the overall performance. If the service volume increases sharply, a master/standby switchover may be triggered.
	Cache breakdown may occur. If Redis cannot handle the pressure on hot keys, requests will hit the database. The database may break down as its load increases dramatically, affecting other services.

14.8.3 How Do I Avoid Big Keys and Hot Keys?

- **Keep the size of Strings within 10 KB** and **the quantity** of Hashes, Lists, Sets, or Zsets **within 5000**.

- When naming keys, use the service name abbreviation as the prefix and do not use special characters such as spaces, line brakes, single or double quotation marks, and other escape characters.
- Do not rely too much on Redis transactions.
- The performance of short connections ("connect" in Redis terminology) is poor. Use clients with connection pools.
- Do not enable data persistence if you use Redis just for caching and can tolerate data loss.
- For details about how to optimize big keys and hot keys, see the following table.

Category	Method
Big key	**Split big keys.** Scenarios: ● **If the big key is a String**, you can split it into several key-value pairs and use **MGET** or a pipeline consisting of multiple **GET** operations to obtain the values. In this way, the pressure of a single operation can be split. For a cluster instance, key-value pairs can be automatically distributed to multiple shards, reducing the impact on a single shard. ● **If the big key contains multiple elements, and the elements must be operated together**, the big key cannot be split. You can remove the big key from Redis and store it on other storage media instead. This scenario should be avoided by design. ● **If the big key contains multiple elements, and only some elements need to be operated each time**, separate the elements. Take a Hash key as an example. Each time you run the **HGET** or **HSET** command, the result of the hash value modulo *N* (customized on the client) determines which key the field falls on. This algorithm is similar to that used for calculating slots in Redis Cluster.
	Store big keys on other storage media. If a big key cannot be split, it is not suitable to be stored in Redis. You can store it on other storage media, and delete the big key from Redis. **CAUTION** Do not use the **DEL** command to delete big keys. Otherwise, Redis may be blocked or even a master/standby switchover may occur.

Category	Method
Hot key	**Use the client cache or local cache.** If you know what keys are frequently used, you can design a two-level cache architecture (client/local cache and remote Redis). Frequently used data is obtained from the local cache first. The local cache and remote cache are updated with data writes at the same time. In this way, the read pressure on frequently accessed data can be separated. This method is costly because it requires changes to the client architecture and code.
	Design a circuit breaker or degradation mechanism. Hot keys can easily result in cache breakdown. During peak hours, requests are passed through to the backend database, causing service avalanche. To ensure availability, the system must have a circuit breaker or degradation mechanism to limit the traffic and degrade services if breakdown occurs.

14.8.4 How Do I Analyze the Hot Keys of a DCS Redis 3.0 Instance?

DCS for Redis 3.0 does not support hot key analysis on the console. Alternatively, you can use the following methods to analyze hot keys:

- Method 1: Analyze the service structure and service implementation to discover possible hot keys.

  For example, hot keys can be easily found in the service code during flash sales or user logins.

  Advantage: Simple and easy to implement.

  Disadvantage: Requires familiarity with the service code. In addition, the analysis become more difficult as the service scenarios become more complex.

- Method 2: Collect key access statistics in the client code to discover hot keys.

  Disadvantage: Requires intrusive code modification.

- Method 3: Capture and analyze packets.

  Advantage: Simple and easy to implement.

14.8.5 How Do I Detect Big Keys and Hot Keys in Advance?

Method	Description
Through **Big Key Analysis** and **Hot Key Analysis** on the DCS console	See **Analyzing Big Keys and Hot Keys**.

Method	Description
By using the **bigkeys** and **hotkeys** options on redis-cli	● redis-cli uses the **bigkeys** option to traverse all keys in a Redis instance and returns the overall key statistics and the biggest key of six data types: Strings, Lists, Hashes, Sets, Zsets, and Streams. The command is **redis-cli -h** *<Instance connection address>* **-p** *<Port number>* **-a** *<Password>* **--bigkeys**. ● In Redis 4.0 and later, you can use the **hotkeys** option to quickly find hot keys in redis-cli. Run this command during service running to find hot keys: **redis-cli -h** *<Instance connection address>* **-p** *<Port number>* **-a** *<Password>* **--hotkeys**. The hot key details can be obtained from the summary part in the returned result.

Hot key analysis is not supported by DCS Redis 3.0 instances. You can **configure alarms** to detect hot keys.

- Configure alarm rules for the **Memory Usage** metric of the instance nodes.

  If a node has a big key, the memory usage of the node is much higher than that of other nodes. In this case, an alarm is triggered to help you find the potentially problematic key.
- Configure alarm rules for the **Maximum Inbound Bandwidth**, **Maximum Outbound Bandwidth**, and **CPU Usage** metrics of the instance nodes.

  If a node has a hot key, the bandwidth and CPU usage of the node is much higher than that of other nodes. In this case, an alarm is triggered to help you find the potentially problematic key.

14.9 Master/Standby Switchover

14.9.1 When Does a Master/Standby Switchover Occur?

A master/standby switchover may occur in the following scenarios:

- A master/standby switchover operation is initiated on the DCS Console.
- If the master node of a master/standby instance fails, a master/standby switchover will be triggered.

  For example, running commands that consume a lot of resources, such as **KEYS** commands, will cause CPU usage to spike and as result triggers a master/standby switchover.
- If you restart a master/standby instance on the DCS console, a master/standby switchover will be triggered.
- If you scale up a single-node or master/standby instance, a master/standby switchover will be triggered.

  During scale-up, a new standby node with the new specifications is created. After full and incremental data on the master node is synchronized to the standby node, a master/standby switchover is performed and the original node is deleted.

After a master/standby switchover occurs, you will receive a notification. Check whether the client services are running properly. If not, check whether the TCP connection is normal and whether it can be re-established after the master/standby switchover to restore the services.

14.9.2 How Does Master/Standby Switchover Affect Services?

If a fault occurs in a master/standby or cluster DCS instance, a failover is triggered automatically. Services may be interrupted for less than half a minute during exception detection and failover.

14.9.3 Does the Client Need to Switch the Connection Address After a Master/Standby Switchover?

No. If the master fails, the standby node will be promoted to master and takes the original IP address.

14.9.4 How Does Redis Master/Standby Replication Work?

Redis master/standby instances are also called master/slave instances. Generally, updates to the master cache node are automatically and asynchronously replicated to the standby cache node. This means that data in the standby cache node may not always be consistent with data in the master cache node. The inconsistency is typically seen when the I/O write speed of the master node is faster than the synchronization speed of the standby node or a network latency occurs between the master and standby nodes. If a failover happens when some data is not yet replicated to the standby node, such data may be lost after the failover.

14.10 Memcached Usage

14.10.1 Can I Dump DCS Memcached Instance Data for Analysis?

No.

14.10.2 What Memcached Version Is Compatible with DCS for Memcached?

DCS for Memcached is based on Redis 3.0.7 and is compatible with Memcached 1.5.1.

14.10.3 What Data Structures Does DCS for Memcached Support?

Only the key-value structure is supported.

14.10.4 Does DCS for Memcached Support Public Access?

No. The ECS that serves as a client and the DCS instance that the client will access must belong to the same VPC. In the application development and debugging phase, you can also use an SSH agent to access DCS instances in the local environment.

14.10.5 Can I Modify Configuration Parameters of DCS Memcached Instances?

Parameter configuration is allowed only when DCS instances are in the **Running** state.

For details, see **Modifying Configuration Parameters**.

14.10.6 What Are the Differences Between DCS for Memcached and Self-Hosted Memcached?

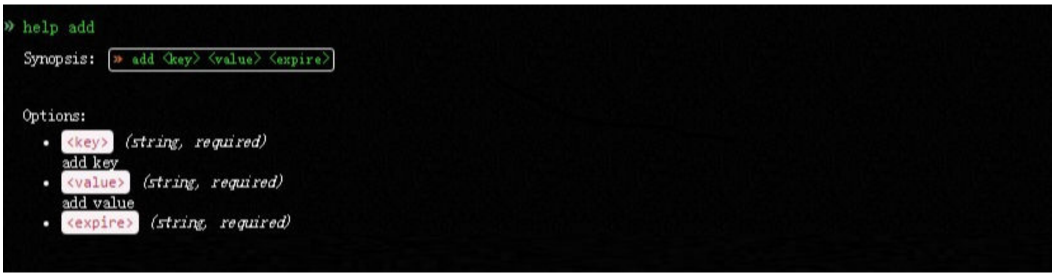
Table 14-11 describes the differences between DCS for Memcached and self-hosted Memcached.

Table 14-11 Comparing DCS for Memcached and self-hosted Memcached

Item	DCS Memcached	Self-Hosted Memcached
Confirming Deployment	Easy to deploy. DCS for Memcached can be used right out of the box without requiring you to worry about the hardware or software.	Involves complicated operations and settings.
Availability	Master/Standby instances use hot standby to ensure stable services. If the master node is faulty, the standby cache node will automatically become the master node to prevent a single point of failure.	Requires additional configurations.
Security	Uses the VPC and security groups for network access security control.	Requires you to design and implement a security mechanism by yourself.
Scale-up	Supports online scale-up on the console.	Requires additional hardware and restarting your service.

14.10.7 What Policies Does DCS for Memcached Use to Deal with Expired Data?

DCS for Memcached allows you to set the expiration time for stored data based on service requirements. For example, you can set the **expire** time when performing the **add** operation.



```
» help add
  Synopsis:  » add <key> <value> <expire>


  Options:
    • <key>  (string, required)
      add key
    • <value>  (string, required)
      add value
    • <expire>  (string, required)
```

By default, data is not evicted from DCS Memcached instances. In the current version of DCS for Memcached, you can select an eviction policy.

For details about the six types of data eviction policies, see **What Is the Default Data Eviction Policy?**

14.10.8 How Should I Select AZs When Creating a DCS Memcached Instance?

Different AZs within a region do not differ in functions.

Generally, instance deployment within an AZ features lower network latency while cross-AZ deployment ensures disaster recovery. If your application requires lower network latency, choose single-AZ deployment.

DCS for Memcached supports cross-AZ deployment. When creating a DCS Memcached instance on the DCS console, you can select any AZ in the same region as your ECS for communication between your ECS and instance. For lower network latency, select the same AZ as your ECS.

Note that there may be only one available AZ due to insufficient resources when you create a DCS Memcached instance. This does not affect the normal use of DCS.

15 Troubleshooting

15.1 Troubleshooting Redis Connection Failures

Overview

This topic describes why Redis connection problems occur and how to solve the problems.

Problem Classification

To troubleshoot abnormal connections to a Redis instance, check the following items:

- **Connection Between Redis and the ECS**
- **Password**
- **Instance Configuration**
- **Client Connections**
- **Bandwidth**
- **Redis Performance**

Connection Between Redis and the ECS

The ECS where the client is located must be in the same VPC as the Redis instance and be able to communicate with the Redis instance.

- For a Redis 3.0 instance, check the security group rules of the instance and the ECS.

  Correctly configure security group rules for the ECS and the Redis instance to allow the Redis instance to be accessed. For details, see **Security Group Configurations**.
- For a DCS Redis 4.0/5.0/6.0 instance, check the whitelist of the instance.

  If the instance has a whitelist, ensure that the client IP address is included in the whitelist. Otherwise, the connection will fail. For details, see **Managing IP Address Whitelist**. If the client IP address has changed, add the new IP address to the whitelist.

- Check the regions of the Redis instance and the ECS.

  If the Redis instance and the ECS are not in the same region, create another Redis instance in the same region as the ECS and migrate data from the old instance to the new instance by referring to the **Data Migration Guide**.
- Check the VPCs of the Redis instance and the ECS.

  Different VPCs cannot communicate with each other. An ECS cannot access a Redis instance if they are in different VPCs. You can establish VPC peering connections to allow the ECS to access the Redis instance across VPCs.

  For details, see "VPC Peering Connection" in the *Virtual Private Cloud User Guide*.

Password

If the instance password is incorrect, the port can still be accessed but the authentication will fail. If you forget the password, you can **reset the password**.

Instance Configuration

If a connection to Redis is rejected, log in to the DCS console, go to the instance details page, and modify the **maxclients** parameter. For details, see **Modifying Configuration Parameters**.

Client Connections

- The connection fails when you use redis-cli to connect to a Redis Cluster instance.

  Solution: Check whether **-c** is added to the connection command. Ensure that the correct connection command is used when connecting to the cluster nodes.
  - Run the following command to connect to a Redis Cluster instance:

    ./redis-cli -h *{dcs_instance_address}* **-p 6379 -a {password} -c**
  - Run the following command to connect to a single-node, master/standby, or Proxy Cluster instance:

    ./redis-cli -h *{dcs_instance_address}* **-p 6379 -a {password}**

  For details, see **Accessing a DCS Redis Instance Through redis-cli**.
- Error "Read timed out" or "Could not get a resource from the pool" occurs.

  Solution:
  - Check if the **KEYS** command has been used. This command consumes a lot of resources and can easily block Redis. Instead, use the **SCAN** command and avoid executing the command frequently.
  - Check if the DCS instance is Redis 3.0. Redis 3.0 uses SATA disks. During AOF persistence, the disk performance may occasionally deteriorate and cause a connection failure. In this case, disable AOF persistence if data persistence is not required. Alternatively, you can use a DCS Redis 4.0 or 5.0 instance because they use SSD disks that offer higher performance.
- Error "unexpected end of stream" occurs and causes service exceptions.

  Solution:
  - Optimize the Jedis connection pool by referring to **Recommended Jedis Parameter Settings**.

  - Check whether there are many big keys. For details, see **How Do I Avoid Big Keys and Hot Keys?**
- The connection is interrupted.

  Solution:

  - Modify the application timeout duration.
  - Optimize the service to avoid slow queries.
  - Replace the **KEYS** command with the **SCAN** command.
- If an error occurs when you use the Jedis connection pool, see **What Should I Do If an Error Is Returned When I Use the Jedis Connection Pool?**

Bandwidth

If the bandwidth reaches the upper limit of the corresponding instance specifications, Redis connections may time out.

You can view the **Flow Control Times** metric to check whether the bandwidth has reached the upper limit.

Then, check whether the instance has big keys and hot keys. If a single key is too large or overloaded, operations on the key may occupy too many bandwidth resources. For details about big keys and hot keys, see **Analyzing Big Keys and Hot Keys**.

Redis Performance

Connections to an instance may become slow or time out if the CPU usage spikes due to resource-consuming commands such as **KEYS**, or too much memory is used because the expiration time is not set for the instance or expired keys remain in the memory. In these cases, do as follows:

- Use the **SCAN** command instead of the **KEYS** command, or disable the **KEYS** command.
- Check the monitoring data and configure alarm rules. For details, see **Configuring Alarm Rules for Critical Metrics**.

  For example, you can view the **Memory Usage** and **Used Memory** metrics to keep track of the instance memory usage, and view the **Connected Clients** metric to determine whether the instance connections limit has been reached.
- Check whether the instance has big keys and hot keys.

  For details about the operations of big key and hot key analysis, see **Analyzing Big Keys and Hot Keys**.

15.2 Troubleshooting High CPU Usage of a DCS Redis Instance

Symptom

The CPU usage of a Redis instance increases dramatically within a short period of time.

Possible Causes

1. The service QPS is high. In this case, refer to **Checking QPS**.
2. Resource-consuming commands, such as **KEYS**, were used. In this case, refer to **Locating and Disabling CPU-Intensive Commands**.
3. Redis rewrite was triggered. In this case, refer to **Checking Redis Rewrite**.

Checking QPS

On the **Cache Manager** page of the DCS console, click an instance to go to the instance details page. On the left menu, choose **Performance Monitoring** and then view the **Ops per Second** metric. If the metric value is high, optimize your service or **increase your instance specifications**. For details about QPS, see **Instance Specifications**.

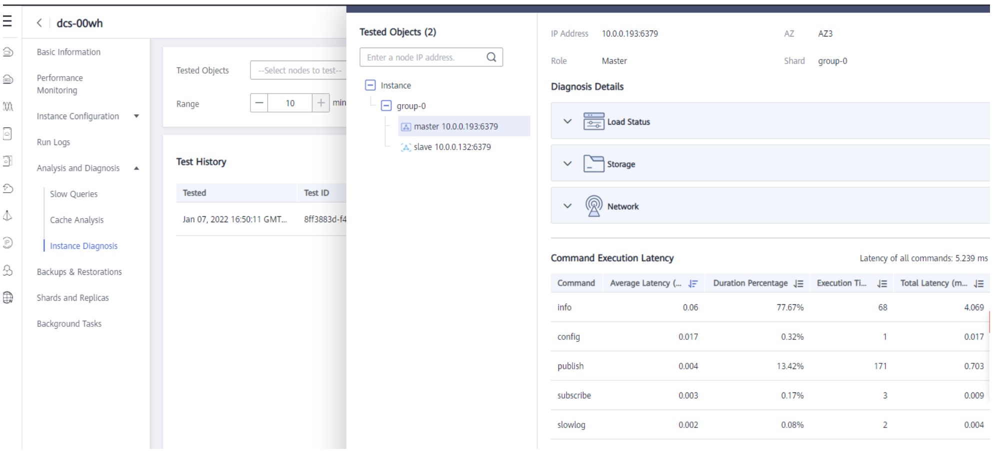
Locating and Disabling CPU-Intensive Commands

Resource-consuming commands (commands with time complexity O(N) or higher), such as **KEYS**, are used. Generally, the higher the time complexity, the more resources a command uses. As a result, the CPU usage is high, and a master/standby switchover can be easily triggered. For details about the time complexity of each command, visit the **Redis official website**. In this case, use the **SCAN** command instead or disable the **KEYS** command.

Step 1 On the **Performance Monitoring** page of the DCS console, locate the period when the CPU usage is high.

Step 2 Use the following methods to find the commands that consume a large number of resources.

- Redis logs queries that exceed a specified execution duration. You can find the commands that consume a large number of resources by analyzing the slow queries and their execution duration. For details, see **Viewing Redis Slow Queries**.
- Use the instance diagnosis function to analyze the execution duration percentage of different commands during the period when the CPU usage is high. For details, see **Diagnosing an Instance**.



Tested	Test ID
Jan 07, 2022 16:50:11 GMT...	8ff3883d-f4

IP Address 10.0.0.193:6379 AZ AZ3
Role Master Shard group-0

Diagnosis Details

Command Execution Latency

Latency of all commands: 5.239 ms

Command	Average Latency (...	Duration Percentage	Execution Ti...	Total Latency (m...
info	0.06	77.67%	68	4.069
config	0.017	0.32%	1	0.017
publish	0.004	13.42%	171	0.703
subscribe	0.003	0.17%	3	0.009
slowlog	0.002	0.08%	2	0.004

Step 3 Resolve the problem.

- Evaluate and disable high-risk and high-consumption commands, such as **FLUSHALL**, **KEYS**, and **HGETALL**.
- Optimize services. For example, avoid frequent data sorting operations.
- (Optional) Perform the following operations to adjust instances based on service requirements:

  Scale up the instance.

----End

Checking Redis Rewrite

AOF persistence, which is enabled by default for master/standby and cluster DCS Redis instances, takes place in the following scenarios:

- If a small amount of data is written and the AOF file is not large, AOF rewrite is performed from 02:00 to 04:00 in the morning every day, and CPU usage may suddenly spike during this period.
- When a large amount of data is written and the AOF file size exceeds the threshold (three to five times the DCS instance capacity), AOF rewrite is automatically triggered in the background regardless of the current time.

Redis rewrite is performed by running the **BGSAVE** or **BGREWRITEAOF** command, which may consume many CPU resources (see **the discussion**). **BGSAVE** and **BGREWRITEAOF** commands need to fork(), resulting in CPU usage spikes within a short period of time.

If persistence is not required, disable it by changing the value of **appendonly** to **no** on the **Instance Configuration** > **Parameters** page of the instance. However, if you disable persistence, data loss may occur due to a lack of data flushing to disk in extreme situations.

15.3 Troubleshooting High Memory Usage of a DCS Redis Instance

Symptom

Redis provides fast database services. If the memory is insufficient, keys may be frequently evicted, the response time may increase, and the QPS may be unstable, affecting service running. This is normal due to Redis functions (such as master/replica replication and lazyfree). When the memory becomes full, scale up the instance or remove unnecessary data. Generally, you need to be alerted when the memory usage exceeds 95%.

Fault Locating

1. Query the memory usage in a specified period. For details, see **Viewing DCS Monitoring Metrics**. Check whether the value of **Memory Usage** is close to 100% continuously.
2. If the values of **Evicted Keys** and **Maximum Command Latency** increase significantly during the period when the memory usage exceeds 95%, the memory is insufficient.

In this case, log in to the console and analyze **big keys** and **slow queries**. If no expiration is set for the instance, too much data will be stored in the instance, using up the memory.

3. If the memory of a Redis instance is full but there are not many keys, the output buffer may have occupied an excessive amount of memory.

   In this case, run the **redis-cli --bigkeys** command to scan for big keys after connecting to the instance using redis-cli. Then, run the **info** command to check the output buffer size.

Solution

- Check for big and hot keys.
  - You can analyze big keys and hot keys on the DCS console. For details, see **Analyzing Big Keys and Hot Keys**.
  - Use the **bigkeys** and **hotkeys** options on redis-cli.
    - redis-cli uses the **bigkeys** option to traverse all keys in a Redis instance and returns the overall key statistics and the biggest key of six data types: Strings, Lists, Hashes, Sets, Zsets, and Streams. The command is **redis-cli -h** *<Instance connection address>* **-p** *<Port number>* **-a** *<Password>* **--bigkeys**.
    - In Redis 4.0 and later, you can use the **hotkeys** option to quickly find hot keys in redis-cli. Run this command during service running to find hot keys: **redis-cli -h** *<Instance connection address>* **-p** *<Port number>* **-a** *<Password>* **--hotkeys**. The hot key details can be obtained from the summary part in the returned result.
- Detect big keys and hot keys in advance.
  - Configure an alarm for node-level memory usage. For details, see **Configuring Alarm Rules for Critical Metrics**.

    If a node has a big key, the memory usage of the node is much higher than that of other nodes. In this case, an alarm is triggered to help you find the problematic key.
  - Configure alarms for node-level **Maximum Inbound Bandwidth**, **Maximum Outbound Bandwidth**, and **CPU Usage**. For details, see **Configuring Alarm Rules for Critical Metrics**.

    If a node has a hot key, the bandwidth and CPU usage of the node is much higher than that of other nodes. In this case, an alarm is triggered to help you find the problematic key.

    NOTE

    You can use the preceding method to detect hot keys for DCS Redis 3.0 instances which do not support hot key analysis.
- Other suggestions:
  - **Keep the size of each String within 10 KB** and **the number of** Hashes, Lists, Sets, or Zsets **within 5000**.
  - When naming keys, use the service name abbreviation as the prefix and do not use special characters such as spaces, line brakes, single or double quotation marks, and other escape characters.

- Do not rely too much on Redis transactions.
- The performance of short connections ("connect" in Redis terminology) is poor. Use clients with connection pools.
- Do not enable data persistence if you use Redis just for caching and can tolerate data loss.

- If the instance memory usage remains high after you take the preceding measures, expand the instance specifications during off-peak hours. For details, see **Modifying DCS Instance Specifications**.

15.4 Troubleshooting High Bandwidth Usage of a DCS Redis Instance

Overview

Redis instances are close to application services, and therefore they process a large amount of data access requests and use network bandwidth. The maximum bandwidth varies depending on instance specifications. If the maximum limit is exceeded, data access performance of application services will be affected. This section describes how to troubleshoot high bandwidth usage of a DCS Redis instance.

Procedure

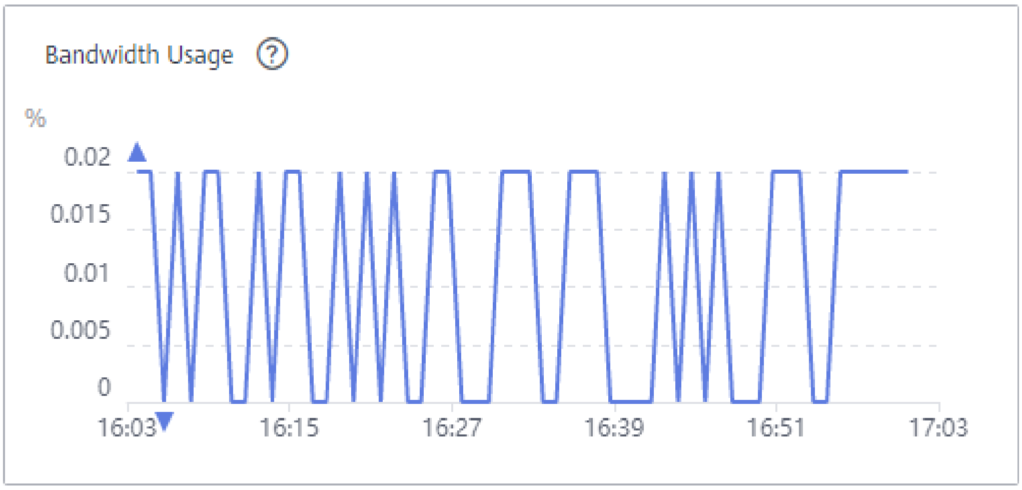
Step 1 Check the bandwidth usage.

Check the bandwidth usage of an instance in a specified period. For details, see **Viewing DCS Monitoring Metrics**.

Generally, if the input and output flows increase rapidly and remain above 80% of the instance's maximum bandwidth, the bandwidth may become insufficient.

The following figure shows the bandwidth usage. Bandwidth usage = (Input flow + Output flow)/(2 x Maximum bandwidth) x 100%

Figure 15-1 Bandwidth usage



Even if the bandwidth usage exceeds 100%, flow control may not necessarily be triggered and can be reflected on the **Flow Control Times** metric.

Even if the bandwidth usage is below 100%, flow control may still be triggered. The real-time bandwidth usage is reported once in every reporting period. Flow controls are checked every second. The traffic may surge within seconds and then fall back between reporting periods. By the time the bandwidth usage is reported, it may have already restored to the normal level.

Step 2 Optimize the bandwidth usage.

1. The service access traffic may not match the expected bandwidth consumption, for example, the bandwidth usage growth trend is inconsistent with the QPS growth trend. If this happens, analyze whether the traffic increase is from read services or write services by checking the input flow and output flow metrics. If the bandwidth usage on a single node increases, use the cache analysis function to detect big keys by referring to **Analyzing Big Keys and Hot Keys**. Optimize big keys (keys larger than 10 KB). For example, split big keys, access big keys less frequently, or delete unnecessary big keys.
2. If the bandwidth usage is still high, scale up the instance to a larger memory size to carry more network traffic. For details, see **Modifying DCS Instance Specifications**.

   NOTE

   Before the scale-up, you can create an instance to test whether the desired specifications can meet the service load requirements. After the test is complete, you can release the instance by referring to **Deleting DCS Instances**.

----End

15.5 Troubleshooting Data Migration Failures

When you use the console to migrate data, the migration may fail if you select an inappropriate migration scheme, the **SYNC** and **PSYNC** commands are not allowed on the source Redis instance, or the network between the source and target Redis instances is disconnected.

This section describes how to troubleshoot data migration failures on the DCS console.

Procedure

Step 1 Check the migration logs.

- If the following error information is displayed, the underlying resources are insufficient to support the migration task. In this case, contact technical support.

  ```
  create migration ecs failed, flavor
  ```

- If the following error information is displayed, the **SYNC** and **PSYNC** commands are disabled on the source Redis instance. In this case, contact technical support to enable the commands.

  ```
  source redis unsupported command: psync
  ```

Step 2 Check whether you used the appropriate migration scheme. If you migrate data between DCS Redis instances, the source instance cannot be a higher version than the target instance.

Step 3 Check whether the **SYNC** and **PSYNC** commands are enabled on the source Redis instance and whether the underlying resources of the migration task are connected to the source and target Redis instances.

This operation is required only for online migration.

For online migration, the source and target Redis instances must be connected, and the **SYNC** and **PSYNC** commands must be enabled on the source Redis instance. Otherwise, the migration will fail.

- Check the network.

  Check whether the source Redis instance, the target Redis instance, and the VMs used for the migration task are in the same VPC. If they are in different VPCs, create a VPC peering connection. For details, see "VPC Peering Connection" in the *Virtual Private Cloud User Guide*.

  If they are in the same VPC, check the security group rules (for DCS Redis 3.0 instances) or whitelists (for DCS Redis 4.0/5.0/6.0 instances) to ensure that the IP addresses and ports of the Redis instances are accessible.

  The source and target Redis instances must be accessible to the underlying VMs used for the migration task. For details about how to configure a security group or whitelist, see **Security Group Configurations** and **Managing IP Address Whitelist**.

  If the source and target Redis instances are on different clouds, create a Direct Connect connection.

- Check the commands.

  By default, the **SYNC** and **PSYNC** commands are disabled by cloud vendors. To enable the commands, contact the O&M personnel of the cloud vendors.

  - Migration within DCS:

    - By default, the **SYNC** and **PSYNC** commands can be used when self-hosted Redis is migrated to DCS.

    - During online migration between DCS instances in the same region under the same account, the **SYNC** and **PSYNC** commands are automatically enabled.

    - During online migration between DCS instances in different regions or under different accounts within a region, the **SYNC** and **PSYNC** commands are not automatically enabled, and online migration on the console cannot be used. You can migrate data using backup files instead.

  - Migration from other cloud vendors to DCS:

    - Generally, cloud vendors disable the **SYNC** and **PSYNC** commands. If you want to use online migration, contact the O&M personnel of the source cloud vendor to enable the commands. For offline migration, you can import backup files.

    - If incremental migration is not required, you can perform full migration by referring to **Online Full Migration of Redis from Another Cloud with redis-shake**. This method does not depend on **SYNC** and **PSYNC**.

Step 4 Check the source Redis instance for big keys.

If the source Redis instances has big keys, split them into small keys before migration.

Step 5 Check the specifications of the target Redis instance and whether other tasks are being performed on the instance.

If the memory of the target Redis instance is smaller than the size of the data to be migrated, the memory will be used up during the migration and the migration will fail.

If a master/standby switchover is being performed on the target Redis instance, contact technical support to stop the master/standby switchover task and start it only after the data migration is completed.

Step 6 Check whether the migration task is performed correctly.

Check whether the IP address and the instance password are correct.

Step 7 Check the whitelist.

Step 8 If the fault persists, contact technical support.

----End

A Change History

Table A-1 Change history

Released On	Description
2024-06-25	Added a description about tag policies in **Creating a DCS Redis Instance** and **Managing Tags**.
2023-09-26	Added description about parameter **active-expire-num** in section **Modifying Configuration Parameters**.
April 3, 2023	This issue is the ninth official release, which incorporates the following changes: ● Added **Notes and Constraints**, **Process of Using DCS**, **Accessing a DCS Memcached Instance (Discontinued)**, **Switching DCS Instance IP Addresses**, **Common DCS Metrics**, and **Data Migration Guide**. ● Added some FAQs and other optimization.
January 3, 2023	This issue is the eighth official release, which incorporates the following change: ● Added description about changing the replica quantity in **Modifying DCS Instance Specifications**. ● Added **Parameter Templates**. ● Added description about Redis 6.0. ● Added **Transmitting DCS Redis Data with Encryption Using SSL**. ● Updated **Creating a DCS Redis Instance**.

Released On	Description
October 19, 2022	This issue is the seventh official release, which incorporates the following changes: Added the following FAQs: **Connection Pool Selection and Recommended Jedis Parameter Settings** **Why Does a Key Disappear in Redis?** **Will Cached Data Be Retained After an Instance Is Restarted?** **How Do I View Current Concurrent Connections and Maximum Connections of a DCS Redis Instance?** **Why Is the Rejected Connections Metric Displayed?** **Why Is Flow Control Triggered? How Do I Handle It?** **Big/Hot Key Analysis** Added **Troubleshooting**.
2022-04-12	This issue is the first official release.